dBASE PLUS 9

User's Guide

Table of Contents

Users Guide

Chapter 1 Intro

Chapter

1

Introduction to *dBASE PLUS*

Welcome to *dBASE PLUS* !

Welcome to dBASE PLUS, the revolutionary, integrated, Information Toolset. Designed from the ground up to provide all the features, functionality and tools required to create and manage the information that fuels today's businesses, dBASE PLUS has something for everyone – from the novice information user to the expert developer.

What is dBASE PLUS?

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

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

With dBASE PLUS you can:

- Work in SQL Server data and save it as Informix
- Work in DB2 and save it as dBASE PLUS
- Tie your legacy systems to your Web Site

- Import data from other applications
- Run reports against almost any database
- Automatically generate applications that work with multiple sources simultaneously.

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

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

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

dBASE Newsgroups

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

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

The dBASE PLUS Knowledgebase:

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

Topics and information include:

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

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

Note

This site can also be accessed through the dBASE PLUS Help menu.

dBASE PLUS documentation

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

Typographical conventions

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

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

Documentation updates and additional information resources

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

Your dBASE PLUS CD also contains a full copy of the new Knowledgebase in HTML format. To use the Knowledgebase, double-click on the file "kbmenu.htm" in the \KB folder on your installation CD. The dBASE PLUS Knowledgebase is also available on the dBase, LLC. website. The Knowledgebase is an ever expanding repository of all things dBASE., check our website frequently for updates!

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

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

Software registration

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

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

To request assistance, call the dBase, LLC. Call Center: 607-729-0960

The call center is open from 8:00 AM to 5:00 PM eastern time USA.

Chapter 2 install

Chapter

2

Installing dBASE PLUS

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

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

What you need to run dBASE PLUS

To run dBASE PLUS you need the following:

HARDWARE

All of the following are required

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

OPERATING SYSTEM

- Microsoft Windows® Vista
- Microsoft Windows® 7
- Microsoft Windows® 8

NETWORKS

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

Products and programs in your dBASE PLUS package

The following products and programs come with dBASE PLUS:

- dBASE PLUS, including integrated compiler/runtime system.
- INNO Setup Quick Start Pack installer (As of the printing of this document current release is 5.4.2).
- DataModule designer.
- The dBASE PLUS debugger.
- The 32-bit Borland Database Engine (BDE) and configuration utility (BDE Administrator), with native drivers for dBASE, Paradox, Microsoft Access 95/97, and Microsoft FoxPro databases.
- Integrated Help system, including a full *Language Reference*.
- The *dBASE PLUS* Knowledgebase.
- Sample tables, forms, reports, and other files that you can learn from, use or adapt.
- A selection of custom controls and graphics (backgrounds, cursors, and other images) for use in forms and reports.
- ODBC connectivity and Local InterBase.
- The SQL Links high-performance drivers allow you to connect directly to Oracle, Sybase, InterBase, MS SQL Server, IBM DB2 and Informix databases.
- ADO components for connecting to 3rd party DBMS databases without the limitations of the BDE

NOTE: dBASE Plus Personal Edition includes all of the files above except the INNO Setup Quick Start Pack installer, the ADO Components and the files needed to build and deploy executables.

Installing dBASE PLUS

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

To install dBASE PLUS,

To install dBASE PLUS, insert the dBASE CD into your CD drive. Or run the Setup.exe from the self-extracting .zip file you downloaded.

IMPORTANT NOTE: The Borland Database Engine will also be installed during the dBASE PLUS installation. *If you have other applications running that use the Borland Database Engine, you must close them before continuing.*

dBASE Plus 9 will NOT install over previous versions of dBASE Plus. A new folder will be created during install … dbase\Plus9. The Shortcuts will be created under START | ALL PROGRAMS | dBASE PLUS 9.

Instructions for each page of the installer: (after the Welcome screen)...
License Agreement : Accept the License agreement and click Next

Destination Folder: Accept the default or choose another folder to install dBASE Plus.

Destination Folder: Accept the default or choose another folder to install the BDE.

Choose Languages: Choose the languages you would like supported by the dBASE Plus IDE.

SetupType: Choose which options to install…
Typical with SQL Links – will install everything!
Typical – will install everything but, the SQL Links

Minimal – will install Just the dBASE Plus IDE and BDE.
Runtime and BDE – will install the Runtime and BDE files.

Ready to Install: Click Next to continue.

After all of the dBASE files have been installed :

1. the BDE Config file (IDAPI.CFG) will be created (if it doesn't already exist).

2. The dbfExplorer app will be installed. This is a utility that allows .dbf file info to be viewed in a Windows Preview window in Windows Explorer

3. Then the third party application "INNO Setup Quick Start Pack" installer will run (unless installing Personal edition).
(see below for details)

4. The final page of the installer will ask if you want to run dBASE PLUS and register it. You can uncheck if you don't want to do this. If you leave this 'checked' and you have not registered yet, the registration dialog will show and you will need to enter your registration information that you received after your purchase.

IMPORTANT NOTE: When the install has completed you might be asked to **re-boot**. THIS IS IMPORTANT as there are some Windows cleanup that is necessary before running dBASE.

Most files are installed under the ***<dBASE Plus root>*** directory.

However, if you are installing under windows Vista, Windows 7 or Windows 8 the default installation for some of these files changes.

Some files (like Samples) are copied to the *<c:\users\<username>\dBase\Plus9…>* directory in order to allow for editing of these files. This is because of the strict control Vista, 7 and 8 implement under most public folders (This is called UAC).
For more information on where files are installed goto *Help | Contents and Index* (in the Plus IDE) and search for *'useUACPaths'* or '*UAC*' under the Index tab.

Install INNO Setup Quick Start Pack Install instructions

–Project Explorer has the ability to generate a script that can be used to create an installation program with Inno Setup.
NOTE: choose 'Cancel' to skip this installation if you already have the most recent version of Inno Setup QuickStart Pack installed.

To install the Inno Setup QuickStart Pack:

- Follow the on-screen prompts.
- Choose installation location – defaults to "C\Program Files\Inno Setup 5"
- Choose Start Menu folder – defaults to "Inno Setup 5"
- Uncheck the box to download and install IS Tool.
- Make sure the box is checked to install Inno

Setup Preprocessor.

- Uncheck the box to download and install encryption support.

What happens during installation

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

- The BFX.OCX and PROJECT.OCX ActiveX controls are installed and registered (Project Explorer controls installed with the dBASE program).
- The dBASE registry settings are written to HKEY_LOCAL_MACHINE\SOFTWARE\dBASE\PLUS.
- In addition to the usual subdirectories, like BIN, a subdirectory named My Projects is created off the dBASE PLUS home directory, by default C:\Program Files\dBASE\Plus\My Projects.
- The Language of the installer will attempt to match the Windows system language setting. This can be set via the Control Panel | Regional Setting. During the install process, you are given the option of selecting additional languages. For example, if you select English and German, the User Interface resources and documentation (as available) will be installed for both languages
- The User Interface language resources installed for the BDE Administrator, the BDE Online Help, and the User Interface resources for the Project Explorer, will match the language of the Installer itself. Multi-language installs are not supported for these components.

What happens when installing on Vista or Windows 7

dBASE PLUS is compatible with Windows 7 and Vista and maintains backward compatibility with Windows XP.

dBASE PLUS adheres to Microsoft's User Account Control (UAC) rules which allow dBASE and dBASE applications to benefit from the enhanced security provided by these rules.

In order to receive the maximum protection provided by UAC, dBASE PLUS defaults to running with standard user rights when installed on Windows 7 | 8 or Vista.

dBASE PLUS contains a number of new features to assist you in working with standard user rights and to simplify upgrading your applications to adhere to UAC as well.

What exactly are the UAC rules from a software developers viewpoint?

Window's UAC rules are intended to:

- Protect installed program files from being modified or damaged by users or programs that should not have access to them.

- Keep each user's files, configuration settings, etc. separate from other users except where shared files or settings are needed.

- Restrict access to any machine wide settings to the maximum extent possible.

By default, only users with Administrator privileges have access to machine wide settings.

Windows 7 and Windows Vista implement these rules by carefully limiting default permissions on folders under the Program Files folder tree, the ProgramData folder tree, the Windows folder tree, and the Users folder tree. Permissions to registry keys are also carefully limited so that standard users will not be allowed to modify any settings that can affect other users.

In order to follow UAC rules a program must:

- place executable code under the Program Files folder tree and NOT attempt to modify or create any new files under this folder tree while running the program. (Standard users generally have read and execute permissions to files under this folder tree. However, programs may be configured to require administrator privileges which would prevent standard users from running them).

- place shared configuration and data files under the *ProgramData* folder tree - but NOT attempt to modify or create any new files under this folder tree while running the program. (By default, standard users have readonly access to this folder tree).

- place master copies of files needed by each user under the *ProgramData* folder tree (to be copied to each user's private folder tree).

- setup a private folder tree under the Users folder tree for each user when a user first runs the program so that each user can modify their private files however they wish without interferring with other users.

How will dBASE PLUS 9support UAC rules?

Installation:

During installation on Windows 8 or Vista, dBASE will install folders containing a default configuration file (plus.ini), sample code, sample application files, and converter utilities under the **C:\ProgramData\dBASE\Plus9** folder.

(Note that on Windows XP, the default for 2.70 will be to continue installing all files for dBASE Plus, as it has in past versions, under the "program files" folder).

In addition, during installation, the Borland Database Engine (BDE) will be configured slightly differently than in the past to place its configuration file under the *ProgramData* tree and set the BDE's NET DIR setting to point to this same folder under *ProgramData*.

Starting dBASE Plus

When dBASE PLUS is started, it ensures that the user has a dBASE specific folder in their *private user folder tree* (i.e. C:\Users\<username>\AppData\Local\dBASE\Plus...) to hold their user specific configuration data (in plus.ini), their private copies of the sample and converter files and any temporary files created while running dBASE.

In addition, the first time dBASE is run by a user it will run a new user setup utility to setup the user's sample and converter files, and generate all the default Source Aliases and default User BDE Aliases.

Also during startup, dBASE will check the user's plus.ini configuration file for any User BDE Aliases and if found, load them into the default BDE session so they will be available for immediate use.
User BDE Aliases are new in dBASE PLUS. They are private BDE Aliases and are stored in each user's plus.ini file. New User BDE Aliases can be created via the Database Wizard accessible via code or via manually editing the plus.ini file. (Note that BDE Aliases can still be created via the BDE Administrator for use by all users on a computer but require Administrator rights to create or delete them).

Un-installing dBASE PLUS

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

Note

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

Chapter 3 Intro to prg

Chapter

3

Introduction to programming in dBL

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

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

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

"Hard coding" vs. visual programming

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

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

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

Advantages of event-driven programs

The three major kinds of user interfaces are

- Command-line, where the user types commands at a prompt. The MS-DOS operating system, the dBASE Dot Prompt (in DOS versions) and Command window (in Windows versions), are examples of command-line interfaces.
- Menu-driven, where the user selects choices from a hierarchy of menu items. Most applications written using prior versions of dBASE provide menu-driven interfaces.
- Event-driven, where the user interacts with visible objects, such as forms containing pushbuttons and list boxes, in any sequence. The user interface is event-driven, and you can create event-driven applications using dBL.

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

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

In a well-designed event-driven application,

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

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

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

How event-driven programs work

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

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

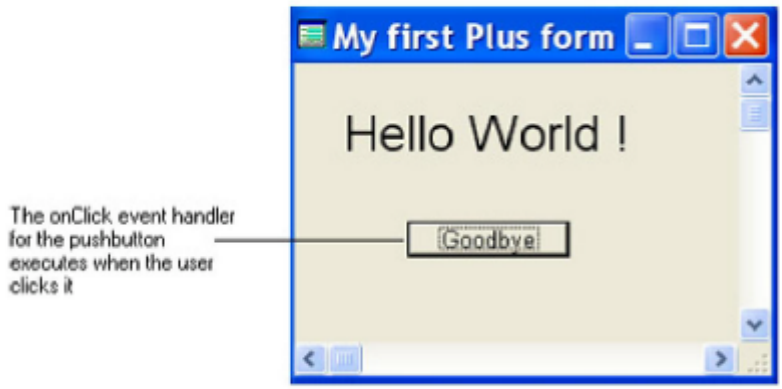
Specifying event handlers for forms is similar to using the ON commands in dBASE DOS, such as ON KEY LABEL or ON ERROR. Like an event handler, the ON command designates some program code to execute when an event, such as a keypress or a run-time error, occurs. However, the events handled by the ON commands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

In a typical event-driven application,

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

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

Figure 0.1 Sample event handler for a "Hello world" form



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

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

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

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

  this.BUTTON1 = new PUSHBUTTON(this)
  With (this.BUTTON1)
     onClick = class::BUTTON1_ONCLICK
     Height = 2
     Left = 19
     Top = 9
     Width = 13
     Text = "Goodbye"
     Metric = 0
```

```
    StatusMessage = "Click button to exit"
    Group = True
  EndWith

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

ENDCLASS
```

Developing event-driven programs

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

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

Chapter 4 Creating and application

Chapter

4

Creating an application

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

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

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

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

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

Creating an application (basic steps)

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

1. If you're creating tables from scratch, plan your tables so you can link them to one another.
2. Plan your directory structure. For example, you may want to put tables in a DATA subdirectory and images in an IMAGES subdirectory.

3. Use the BDE Administrator to create Borland Database Engine (BDE) aliases for all local tables. You can then access those databases through the BDE and through your application. This procedure is described in
4. If several forms (or reports) will be using the same data-source setup (table relationships, SQL statements, methods, and so on) create a data module so you only need to create the data-source setup once. The "Plus data model" and how to access tables using Data Access components is described in Chapter 5 "Accessing and Linking Tables".
5. Create custom form and report classes to give your application a unified look).
6. Create the forms (data entry forms, dialog boxes, and so on) that make up the user interface of your application.
7. Create any reports that your forms will link to or run, using the dBASE PLUS Report wizard and integrated Report designer (see Chapter 6, “Using the Form and Report designers”, and Chapter 11, “Designing reports”).
8. Compile and build your project (choose Build | Compile from the Project Explorer) (in Full version of dBASE PLUS only).
9. Test and debug, using the dBASE PLUS debugger (see Chapter 9, “Debugging applications”).

The Project Explorer

A project (.PRJ) file can be used to help organize and build your application. The project file contains pointers to your application files (tables, forms, queries, bitmaps, and so on). In addition to keeping things organized, having a project file lets you...

- Compile and build a project. (in Full version of dBASE PLUS only)
- Set properties for the project as a whole; for example, you can set compile options, like preprocessor directives.
- Set properties for individual files; for example, you can specify which files you don’t want to include in the build.
- Designate which file should be the first to open when your executable file is run.
- See instant previews of your files in the Project Explorer.
- Open several files at once.

This section discusses the following topics:

- Starting a New Project
- Creating a 'Basic' Application (Non-DEO)
- Creating a 'DEO' Application
- BDE Settings
- INI Files
- ActiveX and DLL Deployment
- Using Project Files From Earlier Versions

Starting a New Project

You can start a New Project by selecting the File | New Project option from the Main menu, or by clicking an Untitled icon on the Project tab of the Navigator. The Project Explorer window is comprised of four pages, accessed by the tabs on the right, and a Treeview along the left side. On the Project Page, you’ll define properties that will determine how, and where, your application will be built.

Adding files to a project

Whether you want to use existing files, or create new ones, the Project Explorer offers several ways to add files to your project.

Adding an existing file:

- Multi-select files from the Add Files dialog by selecting the Project | Add files to project option from the Main menu , or by right-clicking a folder in the Treeview section

Note

Multi-select files by holding down the Ctrl key and clicking on the desired files

- Drag&Drop individual files from the Navigator..

Creating a new file to add to the Project:

- Right-click on the Project Files folder in the Treeview, scroll to the New menu option, and select the desired file type.
- Right-click on any of the other folders in the Treeview and scroll to the New menu option.

In either case, the Project Explorer opens the appropriate designer in which to create the file.

A few things to consider

- If you add tables to a project, you do not need to add the auxiliary files (the .MDX and/or .DBT files in the case of DBFs), just the table itself. The only reason to add a dBASE object file is if you do not have, or do not wish to use, the source code for that file.
- Files that don’t have a designated folder should be placed in the folder marked “Other.”
- Don't forget to add items that are in other libraries. These might include files from the dUFLP code library, Seeker.cc and Report.cc, etc.. You may want to go through your source code and check for references to code in other libraries. You should also check the source code for files you may have from the different libraries, as some of them may have dependencies that are not obvious (for example: :dUFLP:ini.cc depends on code in :dUFLP:StringEx.cc and :dUFLP:SetProc.prg).

Converting Project Files from earlier versions

The current Project Explorer can be used to handle previous generation Project files by making a few basic modifications.

Translating the Project File line

If you use a language other than English as your default, the first line of the file needs to be translated to English. The line should look something like:

Project File generated 03/23/2004 16:25:28

To do this you’ll need to edit this line in the Source Editor:

1. Open the Navigator and select the Project tab
2. Right-click on the project file and select, “Open in Source Editor”.
3. Replace the Project File line with the translated text

Converting the pathnames

All relative paths currently used by your project must be converted to full pathnames.

". . Abandon.bmp"

must be converted to it's full path equivalent

C:\ProgramFiles\dBASE\Plus\Media\Images\Abandon.bmp

Converting relative pathnames:

1. Clicking the filename in the TreeView changes the Project Explorer tab to File Details. If the Project Explorer is unable to locate the file, only the File Name entryfield will display showing the incorrect, or relative pathname.
2. Click the wrench tool next to the pathname.
3. In the Open File dialog, navigate to the file and click Open. The correct pathname and other file information will display in the File Properties section.

You could also modify the pathnames, before bringing the files into the Project Explorer, by using the Source Editor.

Opening a .PRJ file in the dBASE Plus Source Editor:

- Right click on the file in the Navigator, and select "Open in Source Editor" (or press F12)

  or

- In the Command Window, type: MODIFY COMMAND projectname.prj (where "projectname" is the name of your file).

  Entries, such as those below from the [Contents] section, will display in the source code.

```
[Contents]
inventor.wfm,0,0
Library.wfm,0,0
roster.rep,0,0
..\dUFLP\custbutt.cc,0,0
vesper.ini,0,0
..\dUFLP\preview.wfm,0,0
..\Program Files\dBASE\Plus\dBLClasses\FormControls\Seeker.cc,0,0
```

Note

Relative pathnames "..\dUFLP\custbutt.cc,0,0" and "..\dUFLP\preview.wfm,0,0" must be converted to their fullpath equivalents. See "Converting the pathnames" above. Once you have completed this, save the changes you just made and exit the Source Editor (Ctrl+W will accomplish both of these steps at one time).

Opening a Project

To open a project in the Project Explorer, do one of the following:

- Open the Navigator (View | Navigator), select the Project tab, and double click on the project file.
- Select File | Open Project from the Main menu, and navigate to the file in the Open Project dialog.

The Project Page

Project Name

The name you give your project will become the default name for the resulting executable. If, for example, you entered, "MyStuff", as a project name, the name, "MyStuff.exe", will appear as the Target EXE Filename.

DEO Application

Check this box if you want your entire application to be a Dynamic External Object based application. Leave this box unchecked if you only want part, or none, of an application to be DEO. More will be said on DEO later in this section.

Build as UAC App?

This tells the Project Explorer to use the UAC option during the BUILD process. It Indicates whether or not a dBASE application should create and maintain private copies of various files and folders for each user according to Window's User Account Control (UAC) rules.

INI Type

This tells the Project Explorer what type of INI option to use during the BUILD process:
- A Standard INI file (default ini where the ini file is created and saved in the same location as the application)
- A Roaming INI file (an option where the ini file is created and saved in the CSIDL_APPDATA folder)
- or no INI file at all.

Web Application

This tells the Project Explorer to use the WEB option during the BUILD process. It restricts a web application from containing code to create, or use, visual components such as forms, buttons, toolbars, status bars, and other form components, which allows the dBASE Runtime to load faster.

Main Program File

The file that starts your application. This option cannot be set until the project contains at least one program (.prg or .wfm) file.

Main Program Parameters

Used only when you are requiring parameters for startup of the application. Parameters assign data passed from a calling routine to private variables.

Target EXE Filename

The name of the executable resulting from the build process.

Splash Bitmap

The name of the bitmap to be used as the splash screen of your application. This option cannot be set until the project contains a bitmap (.bmp) file.

Program Icon

The name of the icon file to be used as the titlebar of your application. This option cannot be set until the project contains an icon (.ico) file.

Log Filename

A text (.txt) file that lists errors or warnings generated during the BUILD process.

Status

Shows the most recent Date and Time the file was created or modified.

Author

The Project developer.

Description

A name or phrase that serves as the Project identifier.

The File Details Page

To view or edit the details associated with a file, click on it in the Treeview. The Project Explorer opens the File Details page with information displayed about the selected file. The File Details page contains three tabs:

- Details
- Source
- View

The Details tab

File Properties

This section displays the files full pathname, size and dates it was created and last modified

Build Options

This section contains two checkboxes

- “File is Main Startup Program” designates a file as the first to be run when the application is run
- "Include file with executable" tells the Project Explorer whether or not to include the file during the Build process.

  If the file is a Source Code file (Forms, Custom Forms, Reports, Custom Reports, Labels, Programs, Datamodules, Custom Classes): dBASE Plus will compile the file and include it when the .EXE is built.,

  For an Object file: If the file is a dBL object file (a compiled form, etc.), the Project Explorer will build the object file, “as is”, into the .EXE.

  For Other Files (ActiveX, DLL files, header files, etc.): The Project Explorer will, if this checkbox is checked, build (or Bind) the file into the executable. This ensures you will have a non-corrupted file available that a user could obtain through the use of the dBASE COPY command.

Object File Target Location

“Copy File to Separate DEO Folder” designates that a file should to be copied to a folder, and not included in the .EXE during the build process.

- “DEO Folder” is simply the folder where the files will be moved.

The Source tab

For dBL Source code, the Source tab allows you to view the source code for your file. This viewer is read-only. To edit the source code it is necessary to use the Source Editor. To open the file in the Source Editor:

- Right click on the file in the Treeview
- Select the Open in Source Editor option

The Viewer tab

This tab allows you to view your Form, Report, Labels, and Image files, depending on the file type(s).

The Log Page:

This is where you can view the results of any builds in the build log.

The DEO Page:

This is where you can set up DEO locations for your files (see details on DEO applications below).

The INNO Page:

The Defaults tab

Installer name

This field should automatically use the project name indicated under the Project tab. If the name has been changed the Installer name field can be reset to the Project name by clicking on the Refresh button.

Application version

The version number of your project.

Source folder

The root location of the project files.

Installation Directory

The location where the project files will be saved to during the installation process. Level 2 and 3 are optional.

Needed empty subfolder

Specify the name of any empty folders that will be needed within the destination folder when the program is installed. Click the plus sign to add a subfolder, and click the minus sign to remove one.

Setup Language

Specify the language options you want to include for the installation process only. (NOTE: this does not make other languages available in your application).

The Files tab

Destination folder

Set the installation location of the file that is chosen in the source list. Click the button to the right of the field to update the source list.

Support XP, Vista, Windows 7 and Windows 8

Set the security option to use when running the application. You can find this setting in the application's manifest file.

Source

The list of files that will be included in the installation, showing the local project file on the left and the installation path on the right

Flags

Click this button to set various attributes for each file.

Flags Parameter (Files tab)

This page displays options for the Inno script [Files] section Flags Parameter. Selected options will be applied to the current file. To select an option, click in the checkbox to it's left. To access the flag options page, click the "Flags" button located in the bottom right-hand corner of the Files page. See Inno Script Generator Help for additional details on Inno Script Generator program options.

The following options are supported:

CompareTimeStamp (Special-purpose)

Instructs Setup to proceed to comparing time stamps.

When a file being copied already exists on the user's system, and has the same version info, selecting this option.instructs Setup to overwrite the existing file only if it has an older time stamp than the version of the file Setup is trying to install. When this option is left unchecked, Setup would not try to overwrite the existing file. This flag has no effect if the copy mode isn't either normal or AlwaysSkipIfSameOrOlder.
This flag is left in for backward compatibility only, and we recommend that you not use it.

- NT users may encounter false "existing file is newer" messages when they change their system's time zone, or when daylight savings time goes into effect.
- A similar problem can occur if an installation was compiled on an NTFS partition and the files are installed to a FAT partition, because times only have a 2-second resolution.on FAT partitions

ConfirmOverWrite

Instructs Setup to ask for user confirmation before overwriting an existing file.

DeleteAfterInstall

Instructs Setup to copy the file as usual, but delete it once the installation is completed or aborted. This can be useful for extracting temporary data needed by a program executed in the script's [Run] section.

This flag will have no affect on existing files that weren't replaced during installation.

This flag cannot be combined with the IsReadMe, RegServer, RegTypeLlib, RestartReplace, SharedFile, or UninsNeverUninstall flags.

RegServer

Instructs Setup to register the OLE server (a.k.a. ActiveX control), by locating and executing the DLL/OCX's DllRegisterServer export. The uninstaller calls DllUnregisterServer. When used in combination with SharedFile, the DLL/OCX will be unregistered only when the reference count reaches zero.

See the Remarks at the bottom of this topic for more information.

RegTypeLib

Instructs Setup to register the type library (.tlb). The uninstaller will unregister the type library (unless the flag UninsNeverUninstall is specified). When used in combination with SharedFile, the file will be unregistered by the uninstaller only when the reference count reaches zero.

See the Remarks at the bottom of this topic for more information.

NoRegError

When used in combination with either the RegServer or RegTypeLib flags, instructs Setup not to display an error message if the registration fails.

PromptIfOlder
When a file being installed has an older version number (or older time stamp, when the CompareTimeStamp flag is used) than an existing file, instructs Setup to give the user the option to replace the existing file. See the Remarks section at the bottom of this topic for more details.

RestartReplace
Instructs Setup to register locked files, designated for replacement, in either WININIT.INI or MOveFileEx for Windows and Windows NT respectively. These files will be replaced during the next system startup. When such files are encountered, the user will be prompted to restart the computer at the end of installation.

This flag is generally useful when replacing core system files.

To maintain compatibility with Windows 95/98 and Me, long filenames should not be used on an entry with this flag. Only "8.3" filenames are supported. (Windows NT platforms do not have this limitation.)

Important:

The RestartReplace flag will only successfully replace an "in-use" file on Windows NT platforms when the user has administrative privileges. If the user does not have administrative privileges, the following message will be displayed;

"RestartReplace failed: MoveFileEx failed; code 5."

Therefore, when using RestartReplace it is highly recommended that your installation require administrative privileges, by setting "PrivilegesRequired=admin" in the [Setup] section.

UnInsRestartDelete
Instructs the uninstaller to queue a file, currently in use, to be deleted when the system is restarted. When such files are encountered, the user will be prompted to restart the computer at the end of uninstallation.

This flag can be useful when uninstalling files such as shell extensions which cannot be programmatically stopped.

Note:

Administrative privileges are required on Windows NT/2000/XP for this flag to have an effect.

UnInsNeverUninstall
Instructs the uninstaller to never uninstall this file. Never remove the file. This flag can be useful when installing very common shared files that shouldn't be deleted under any circumstances, such as MFC DLLs.

Note:

If this flag is combined with the SharedFile flag, the file will never be deleted at uninstall time but the reference count will still be properly decremented.

UnInsRemoveReadOnly
Instructs Setup to remove a files' read-only attribute before attempting to delete it. during uninstallation

RecurseSubDirs
Instructs the compiler to search for the Source <filename><wildcard> in the subdirectories as well as the Source directory.

CreateAllSubDirs
Instructs Setup to create subdirectories, associated with a file, when they don't currently exist. Has an effect only when used in combination with RecurseSubDirs.

SortFilesByExtension
Instructs the compiler to compress the found files, sorted by extension before being sorted by path name. This potentially decreases the size of Setup if SolidCompression is also used.

FontIsntTrueType
Specify this flag if the entry is installing a non-TrueType font with the FontInstall parameter.

IsReadMe
Instructs Setup that this is the "README" file. Only one file in an installation can have this flag. When a file is designated a README, users will be given the option to view the README file after installation has been completed. If so desired, Setup will open the file with the default program the user has designated for that file type. For this reason, the README file should always end with an extension like .txt, .wri, or .doc.

Note:

The user will not be given an option to view the README file.when Setup has been instructed to restart the computer (as a result of installing a file with the flag RestartReplace, or if the AlwaysRestart [Setup] section directive is yes).

External
Instructs Setup to copy the file, specified by the Source parameter, from an existing file on the distribution media, or the user's system. If left unchecked, Setup will, instead, statically compile the file into the installation files.

SkipIfSourceDoesntExist
Instructs the installer, or Setup, if the external flag is also used, to silently skip over an entry, and not display an error message, when the source file does not exist. Has effect only when used in combination with the External flag.

OverWriteReadOnly
Instructs Setup to always overwrite a read-only file. Without this flag, Setup will give the user the option to overwrite existing read-only files.

OnlyIfDestFileExists
Instructs Setup to copy a file only when a file of the same name exists on the user's system. This flag may be useful if your installation is a patch to an existing installation, and you only want to replace files currently residing on the user's system.

OnlyIfDoesntExist
Instructs Setup to install a file only when it does not currently reside on the user's system.

ReplaceSameVersion
Instructs Setup to replace current files even when they appear to be the same version as the newer files. The default behavior is to retain the current files.

SharedFile (Windows 95/NT 4+ only)
This flag uses Windows' shared file counting feature, located in the registry at:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs)

It enables a file to be shared between applications, without worrying about it being inadvertently removed. Each time the file is installed, the reference count for the file is incremented. When an application using the file is uninstalled, the reference count is decremented. If the count reaches zero, the file is deleted (with the user's confirmation, unless the UnInsNoSharedFilePrompt flag is also specified).

Most files installed to the Windows System directory should use this flag, including .OCX, BPL, and .DPL (Delphi 3 package) files. One of the few exceptions is MFC DLLs, which should use the OnlyIfDoesntExist copy mode, in conjunction with the UninsNeverUninstall flag.

UnInsNoSharedFilePrompt
Instructs Setup to automatically remove a shared file, during uninstallation, whose reference count has reached zero. No user prompt will be displayed.

This flag must be used in combination with the SharedFile flag to have an effect.

NoEncryption
Prevents the file from being stored encrypted.

NoCompression
Prevents the compiler from attempting to compress the file.

DontVerifyChecksum

Prevents Setup from attempting to verify the checksum of the file. This flag must be used in combination with the NoCompression flag to have an effect.

Touch
Instructs Setup to set the time/date stamp of the installed file(s) to that which is specified by the TouchDate and TouchTime [Setup] section directives.

This flag has no effect if combined with the external flag.

Remarks
Setup registers all files with the RegServer or RegTypeLib flags as the last step of installation. However, if the [Setup] section directive AlwaysRestart is yes, or if there are files with the RestartReplace flag, all files get registered on the next reboot (by creating an entry in Windows' RunOnce registry key).

When files with a .HLP extension (Windows help files) are uninstalled, the corresponding .GID and .FTS files are automatically deleted as well.

DontCopy
Setup will not copy the file to the user's system. This flag is useful if the file is handled by the [Code] section exclusively.

IgnoreVersion
Setup will not compare any version info, and existing files will be overwritten regardless of their version number. This flag should only be used on files private to your application - never on shared system files.

The Menu Group tab

Menu Group File

The file to show in the Start Menu.

Start Menu Group

Where to show the application in the Start Menu.

Menu Label

Name of the Start Menu item.

Existing Menu Group Items

List of the current Start Menu items, with the Start Menu location on the right and local location on the right.

Support Desktop Icon

Whether or not to include an option to install a Destop Icon.

Support Quicklaunch Icon

Whether or not to include an option to install a Quicklaunch Icon.

Include uninstall icon

Indicate if the user should be able to uninstall the application from the Start Menu

Include program to modify, repair or remove installation

When checked, a program maintenance application is installed and listed in the Start Menu that allows the user to modify, repair or remove the installation.

Display group window at end of setup

Choose this option to open the Start Menu folder when the installation completes.

The Runtime tab

Include Runtime Engine

Includes the dBASE Runtime Engine in the installation. Options such as language, progress indicators and destination folder can be set up in this section.

Include Runtime Files

Use this option if an update to the application is being made, not requiring the BDE and Runtime Engine (which gets installed when using Runtime Engine option above) to be included. Indicate the language and destination folder for the Runtime files. You can also choose to automatically include a manifest file with either 'requireAdministrator', 'Highest' or 'asInvoker' status.

Do not include runtime

When selected, the dBASE Runtime will not be included in the installation.

The License tab

Language

Select a language for the text files. Files can be set for more than one language.

License file

Choose a text file that contains the License Agreement.

Text before installation begins

Choose a text file that contains any text or messages that should display in the installation window before the installation begins.

Text after installation completes

Choose a text file that contains any text or messages that should display in the installation window after the installation completes.

Selected license files

Lists the files that are currently set for the license and messages during the installation

The BDE Settings tab

Folder

The root folder in the installation where the database alias should link to. Use the dropdown to choose {app} for the application root folder. If a subfolder is needed, choose {app}\<sub_folder>.

Name

The name of the alias to include.

Driver

The database driver that should be used..

Delete before create

If checked, an existing alias with the specified name will be removed before setting up the new database alias.

Database aliases to create at install time

The list of aliases that will be created during the installation. To add an alias, click the plus sign; to remove an alias, click the minus sign.

Update BDE Settings

Set defaults for: MAXFILEHANDLES; LANGDRIVER; dBASE table LEVEL; MEMO FILE BLOCK SIZE; and MDX BLOCK SIZE.

The INI tab

Include default INI entries

Includes an INI file containing default entries.

Include error handling

Includes the INI entry for error handling.

Trap all errors: Includes the INI error handling option to trap all errors

Ignore interpreter memory access violations: Includes the INI error handling option to ignore interpreter memory access violations.

BDE not required

Check this box to include an INI file entry which indicates the Borland Database Engine (BDE) is not required by this application. When an INI file contains this entry, the application will run without trying to load the BDE, making it unnecessary to install the BDE on the target system

The remaining fields will set the BDE driver properties. The fields are initially set to the default BDE dBASE driver values

UAC Registry Setting tab

To specify whether or not installer creates an application specific registry setting for useUACPaths. During the installation of your application you can decide whether or not the installer will create a useUACPaths setting for the <app>.exe that you create.

These options are available:

Do not create application specific registry setting for useUACPaths.

This will allow any embedded UAC setting on the application .exe or the Runtime registry uac setting to decide what happens with UAC control

Create application specific registry setting with useUACPaths="Y"

this will override both the runtime registry setting and the embedded UAC setting on the .exe

Create application specific registry setting with useUACPaths="N" on Windows Vista or newer.

this will override both the runtime registry setting and the embedded UAC setting on the .exe

When your application is run, the option to use UAC paths is determined in this manner...

If a useUACPaths registry key exists for dBASE Plus, its used to set the default for app.useUACPaths. If this key does NOT exist, _app.useUACPaths is defaulted to FALSE.

If an application .exe is built with an embedded UAC setting, the embedded setting will override the global default set for the dBASE Plus IDE.

If an application specific registry key exists, its setting will override the dBASE Plus registry key setting and the embedded UAC setting in the application .exe (if one exists).

If a -v switch is passed on the command line, it will override all of the above settings.

The Script tab

To specify the name of the script, click on the folder icon. The script should have an .iss extension.

To generate and save the script, click on the disk icon. The script can then be seen in the large box below the icons. The text line above the script will indicate if the script is current with the latest settings or not.

Once the script and project exe are generated, the installer can be created by launching Inno Script Generator with the lightning bolt icon. Once Inno Script Generator is open, create the installer by clicking on the Inno button under compile. By default, the installer will be in a folder called Output, located in the source directory

Creating an Application

Once all the necessary files have been included in a project, creating a dBASE application is quite simple using the Project Explorer.

- Complete the Project page
- Add your files
- Designate a Main Start program by checking the, “File is Main Startup Program?”, box in the Build Options section of the file’s Detail tab.
- Compile and Build the Application

Creating a DEO Application

- Check the box marked, “DEO Application?”, on the Project page.
- Click “Yes” on the ensuing dialog.

After designating a project as a DEO application, you need to assign the project files to DEO folders.

DEO Folders

1. Select the DEO tab
2. If you have existing folders you wish to use for your DEO deployment, you can select them by clicking on the 'folder' icon. This will open the Choose Directory dialog, allowing you to browse to a folder. Once you have located the desired folder, click OK and the file will be added to the DEO folder list.
3. If you wish to create new folders, type the full path into the entryfield and press the ENTER key. The Project Explorer will create the folder.
4. Repeat this for each folder you wish to create by selecting the next numbered line in the listbox, and designating a path and name in the entryfield.
5. To remove a folder from the list you must first remove any files assigned to it. Once it is empty, select it from the list and click the “Clear a Target Folder” icon on the right.Selecting the DEO Folder For Each File

Selecting the DEO Folder For Each File

Once you have created the DEO folders we need to designate which files the Project Explorer will place in the various folders.

1 In the TreeView, select the file you want assigned to a particular folder. The Project Explorer switches the view to the Source or Details tab of the File Details page.

2 Make sure the Details tab is selected and Uncheck the, “Include file within executable”, box

3 Check the,” Copy File to Separate DEO Folder", box.

4 Repeat this process for each file that can be compiled.

The Project Explorer will not allow you to set the DEO location for a file you have designated as the Main Startup Program in the Build section. The application needs that file compiled and built into the executable itself.

Building the Executable

Select Build | Build or Build | Rebuild All from the Project Explorer’s Main menu.

- Build assumes that all of your files have been compiled, and just builds the executable. It will return an error if an object file cannot be found.
- Rebuild All recompiles all compilable files. If you designated files to be moved to a DEO folder in the Object Folder File Location section of their File Details, the Project Explorer copies the appropriate files to their designated folders, and builds the .EXE file.

Errors occurring during the build process will result in an “Error Message” that will be written to the project’s Logfile you designated on the Project page.

Once a build has been successfully completed, you can run the .EXE by navigating to it with Windows Explorer or by selecting the Build | Execute (filename) option from the Project Explorer’s Main menu.

Suggestion

If you run your executable in the same folder that contains your source code, and you have made changes that have yet to be built (bound) into the executable, add the following line to the startup code for the main program:

```
_app.allowDEOExeOverride := false
```

This will prevent dBASE Plus from assuming this is a DEO Application. Basically, this line tells the dBASE Plus runtime to reverse the order normally used when looking for object files. For more information on DEO, see the topics Dynamic External Object (DEO) and allowDEOExeOverride in Help .

.INI files

To insure your deployed application performs as intended, you can deploy an .INI file of the same name as the .EXE.

Some settings you should consider (see online help in dBASE for details for SET EPOCH, SET CENTURY, SET LDCHECK, etc.) adding as an example below:

```
[CommandSettings]
EPOCH=1950
LDRIVER=WINDOWS

[OnOffCommandSettings]
CENTURY=ON
LDCHECK=OFF
BELL=OFF
TALK=OFF

[CommandWindow]
Open=1
Maximized=0
Minimized=1

[ComponentTypes]
ComponentTypeNumeric=1
ComponentTypeDate=1
ComponentTypeLogical=0
ComponentTypeMemo=0
```

```
ComponentTypeBinary=0
ComponentTypeOLE=0
ComponentTypeNumericForTables=1
ComponentTypeDateForTables=1
ComponentTypeLogicalForTables=0
ComponentTypeMemoForTables=0
ComponentTypeBinaryForTables=0
ComponentTypeOLEForTables=0

[Grid]
DefaultMemoEditor=0

[Toolbars]
Format=0

[ObjectPath]
objPath0=c:\path
objPath9=c:\anotherpath

[IDAPI]
CONFIGFILE01=mycustom.cfg
```

The **LDRIVER=WINDOWS** setting ensures that no matter what your application's BDE Driver, your source code will be saved as ANSI.

Setting **TALK**=ON will cause dBASE Plus to constantly echo commands to the command window, and may cause performance degradation. In dBASE Plus, the Runtime Engine automatically assumes talk is OFF

The **ComponentTypes** settings reduce the likelyhood of datatype mismatches, particularly if you are using the grid component on your forms. You should copy the section shown above from your own PLUS.INI, as you may have different settings than those shown above.

Grid was new to dBASE Plus in version 2.21, and is used to set the default columnEditor type for memo fields in a grid. If DefaultMemoEditor is set to zero, the default (columnEditor) is used, if set to one (1), the columnEntryfield is used.

Toolbars: When the Format option is set to "1", the Format Palette is displayed when a form with an editor control is opened. The Format Palette does not display when this is set to “0”.

ObjectPath: This is how DEO is handled. Briefly, when you run an executable built with dBASE, it checks:

1. It looks in the "home" folder from which the application was launched.
2. It looks in the .ini file to see if there's a series of search paths specified. It checks all the paths in the list looking for the object file requested by the application.
3. It looks inside the application's .exe file, the way Visual dBASE did in the past.

IDAPI is only really necessary if you are using a custom configuration file for the BDE. This may cause a problem if multiple programs on the same computer try to use the BDE with different configuration files. It is recommended that other methods of modifying the BDE's setup are used, such as running code in the dBASE Users' Function Library (dUFLP) that will modify the BDE’s registry settings.

Encrypted Tables

If you are working with tables encrypted using the dBASE PROTECT command, you will need to:

- Deploy the DBSYSTEM.DB file
- Have an entry in your .INI that looks like:

  ```
  [CommandSettings]
  DBSYSTEM=D:\PATH
  ```

  Where, D:\PATH, is the path to the directory where you are deploying this file.

If you already have a "[CommandSettings]" section, just add the "DBSYSTEM" entry.

If you are deploying the DBSYSTEM file to the same as your application, set this to "."

```
DBSYSTEM= "."    //the "." stands for "current directory"
```

OCX and DLL controls

OCX and some DLL Controls require some registry settings. You can make these with your install software, or you can code them in your own application.

Using Inno

To make an installer for a project, use the Inno tab on the Project Explorer to create an installation script. Using Inno Setup and Inno Script Generator, the generated script can then be used to create the installer. This section discusses the basic steps of creating an installer using the Project Explorer. It includes information on:

- Setting defaults
- Including files
- Windows Start Menu settings
- Including the Runtime
- Setup a License agreement
- Including the BDE and required aliases
- Naming the Script

Building the user interface

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

Components contain three kinds of information:

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

Form design guidelines

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

Goal of form design

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

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

Purpose of a form

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

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

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

Some guidelines for data entry forms

When designing data entry forms, consider the following guidelines:

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

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

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

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

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

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

Designing the form layout

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

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

Guidelines for using the z-order

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

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

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

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

Creating a form

You can create a form in two ways:

1. **Use the Form wizard.** The Form wizard creates a data-entry form. It presents you with a series of options, and based on your selections, creates a form It saves time, and you can modify and further develop the form in the Form designer (see "Using the Form wizard' on page 76).
2. **Use the Form designer.** Here you can create a form from scratch, by dragging components onto the form and specifying their properties. Since components have built-in functionality, you can actually create very simple applications with little or no coding. However, to create more complex and highly customized applications, you need to write your own event handlers and methods for various components.

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

Using the Form wizard

To use the Form wizard,

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

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

Using the Form designer

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

1. Place components on the form. To do so, drag files (including data modules, if you're using them) from the Navigator or Windows Explorer to automatically link a table or database to a form, and drag the objects you need from the Component and Field palettes onto the design surface.
   Note: If you drag tables onto the form, the fields that are available on the Field palette are already linked to data. To link any other component to a field, set its *dataLink* property.
2. Set component properties, using the Inspector (or the Source editor, if you prefer).
3. Attach code to component events and write the methods you need.
4. Create menus, as necessary, using the Menu designer (see Chapter 7, "Creating menus and toolbars").
5. Create toolbars and tool buttons, as necessary (see Chapter 7, "Creating menus and toolbars").

The Form designer creates a .WFM file.

The components available in dBASE PLUS and the mechanics of using the Form designer, including the Inspector, are discussed in Chapter 6, "Using the Form and Report designers". Also see the samples that come with dBASE PLUS, installed by default in the Plus\Plus\Samples directory.

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

.WFM file structure

The following code was generated by

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

Here is the code:

```
** END HEADER -- do not remove this line
*
* Generated on 08/24/97
*
parameter bModal
local f
f = new UntitledForm( )
if (bModal)
  f.mdi = .F. && ensure not MDI
  f.ReadModal( )
else
  f.Open( )
endif

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

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

this.Rowset = this.animals1.Rowset

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

ENDCLASS
```

There are four major sections in a .WFM file:

1. The first part is the optional Header section. This is any code above the ** END HEADER line. Comments that describe the file are usually put here.

2. Between the header and the beginning of the CLASS definition is the standard bootstrap code. This code instantiates and opens a form when you run the form, similar to the way the boot sector of a disk starts the system when you turn on your computer. The standard bootstrap code allows you to open the form in two ways:
   - If you DO the .WFM with no parameters, the form is opened with the *open*( ) method. The form is modeless.
   - If you DO the .WFM with the parameter True, the form is opened with the *readModal*( ) method. The form is modal. A modal form cannot be MDI, so the form's MDI property is set to False first.
3. The main CLASS definition constitutes the bulk of most .WFM files. This is the code representation of forms designed visually in the Form designer.
4. Everything after the main class definition, if anything, makes up the General section. This is a place for other functions and classes.

Form class definition

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

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

How the contents are generated

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

Editing a .WFM file

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

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

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

Editing the header and bootstrap

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

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

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

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

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

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

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

Editing properties in the .WFM file

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

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

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

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

```
value = date( )
```

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

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

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

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

Types of form windows

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

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

The following sections briefly describe these form types.

MDI and SDI applications

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

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

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

Modal and modeless windows

dBASE PLUS lets you create both modal and modeless windows.

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

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

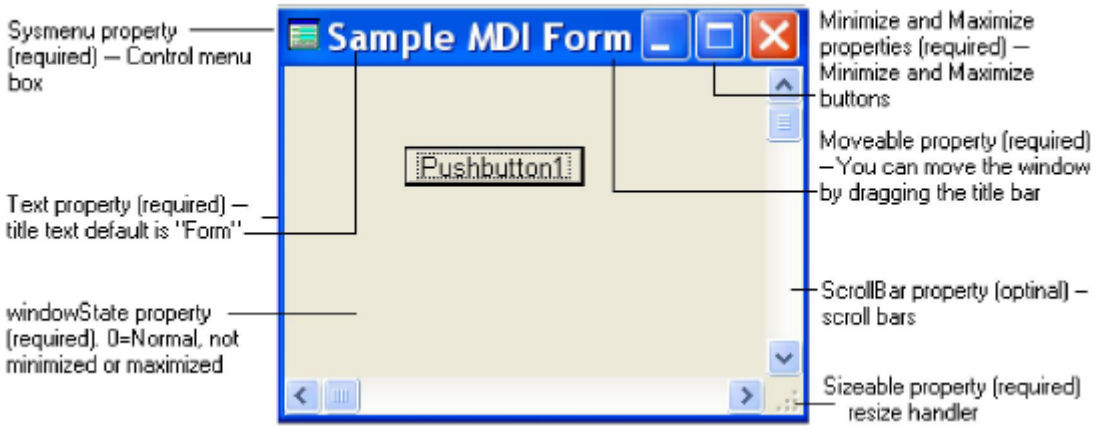
Customizing the MDI form window

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

Standard features of MDI windows:

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

Figure 0.1 Sample MDI window



If the MDI property is set to *true*, those features are automatically applied to the form. Accordingly, the following form properties are automatically set to *true*: *Minimize*, *Maximize*, *Sizeable*, *Moveable*, and *SysMenu*. Changing the

MDI-required properties will have no effect until you change the MDI property itself to *false*. For more information about any of these properties, press F1 when the property in the Inspector is highlighted.

Using multi-page forms

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

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

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

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

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

Global page (forms)

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

To open page 0,

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

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

Important

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

Navigation buttons (form pages)

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

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

1. Go to the global page, page 0 of the form (View | Go To Form Page Number).
2. Select the form itself in the Inspector's top selection box, and make sure the *pageNo* property is 0.
3. From the Component palette drag as many button components as you need to the visual design surface. Ensure that the buttons will not overlap controls appearing on other pages.
4. Select each button and set its *pageNo* property to 0 (the global page) so that it will appear on all pages. (Or multi-select the buttons and set the property once.)
5. Select each button's *text* property and change its value to whatever you want on the button, for example Next Page or Previous Page.

6. For each button, on its Events page, select *onClick* and click the tool button to display the Source editor. You'll see a comment for an *onClick* method. Write the code that will send the user to the appropriate page. Return to page 1 before running the form.

Creating a custom form, report, or data module class

When you use the designers in dBASE PLUS, by default the Form designer uses the Form class, the Report designer uses the Report class, and the DataModule Designer uses the DataModule class.

However, you can create *custom classes* and use them as templates (both for new forms, reports, and data modules and those already created). For example, if you want many forms in your application to have a similar look, you can specify all the common attributes for those forms, such as colors, size, controls, event handlers, and so forth, once. When you have established all the common attributes, save that form as a custom form class. Then you can specify that class to be a template for forms. Changes you subsequently make to the custom form class will be reflected in all its derived forms.

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

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

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

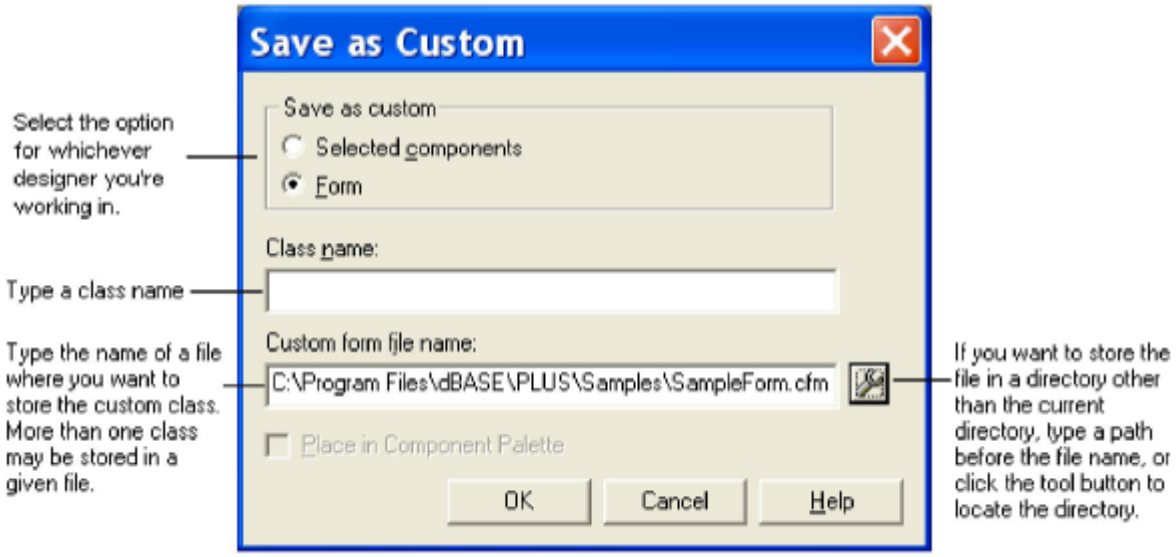
An alternate way to create a custom class is to double-click the yellow, "Untitled" icon on the Forms, Reports, or Data Modules page of the Navigator. This opens the appropriate Custom Class designer, which is almost identical to the Form or Report designers. Then add the common features you want to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

Note

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

.

Figure 0.1 Saving a custom class



Using a custom class

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

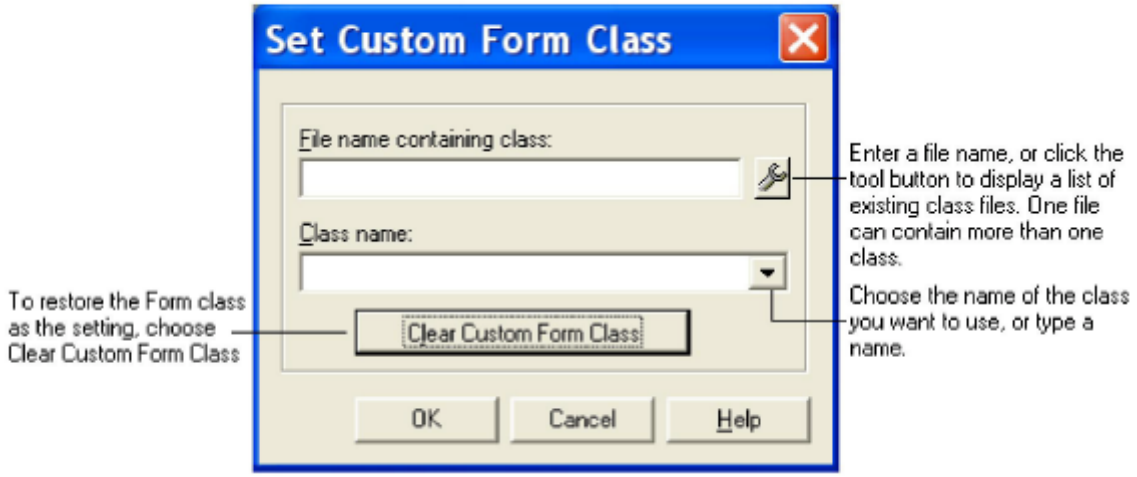
1. Open a new or existing form or data module to which you want to apply a custom class.
   - Setting a custom report always causes a new report to be created. To apply a custom class to an existing report, open the report in the source code editor and change the CLASS statement:

     CLASS MyReport OF Report

     to read:

     CLASS MyReport OF "MyCustomReport" FROM "MyCustomRep.CRP"
2. From the designer, choose File | Set Custom Form (or Report or Data Module) Class.
3. Complete the Set Custom Class dialog box and choose OK.

Figure 0.1 Set Custom form Class Dialog Box



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

Creating custom components

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

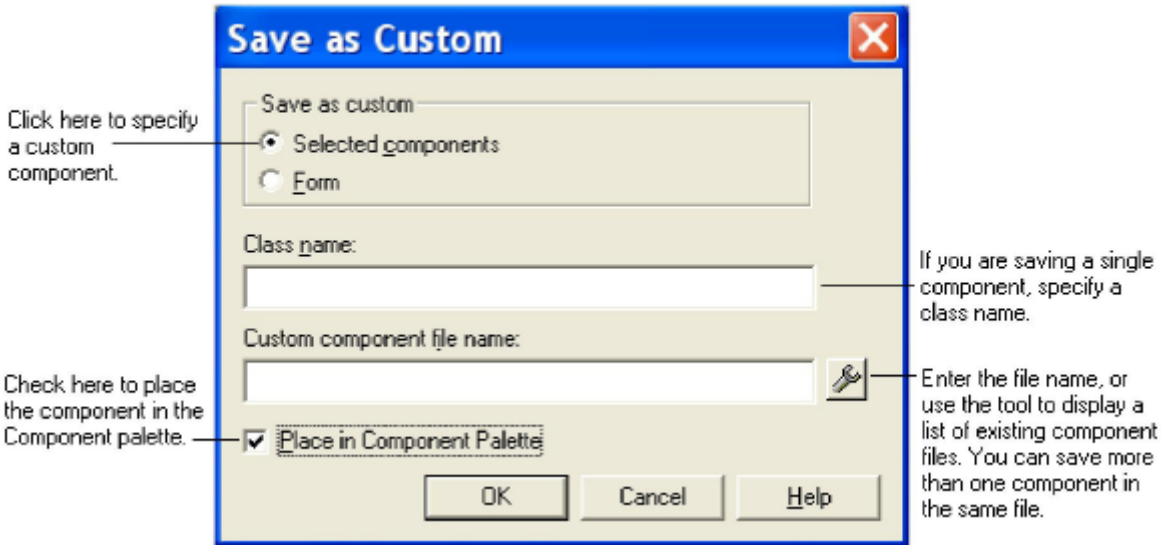
Then you save your work into a custom component file (with the .CC extension) and optionally add it to the Component palette for convenient access.

To create custom components that you can use again,

1. Drag a component or components to the Form or Report design surface, and arrange them the way you want them.
2. Set each component's properties, events, and methods.
3. Select the component or group of components.
4. Choose File | Save As Custom to display the Save As Custom dialog box, then complete the dialog box:

- Type a class name for the customized component.
- Specify an entire path name and the file (with the .CC extension) in which you want to store this component. Note that the components in a .CC file are treated as a group, and although you can add them to the palette individually in this dialog box (by checking the appropriate check box), you can remove them from the palette only as a group. So, if you want to be able to add and remove custom components from the palette separately, put each in its own file.
- Check the Place In Component Palette check box, if you want this component to appear on the Component palette. If you are putting the component in an existing .CC file whose components are already on the palette, and you don't check this box, then later, if you want to add the component to the palette, you'll have to remove the .CC file from the Set Up Custom Components dialog box, and then add it anew.

Figure 0.1 Save as Custom dialog box; saving Custom Components



5. Click OK.

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

Adding custom components to the Component palette

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

1. Choose File | Set Up Custom Components (or right-click in the Component palette for access to the same command).
2. The Set Up Custom Components dialog box appears. It lists paths to custom component files whose components already appear on the Component palette.
3. Choose Add to open the Choose Custom Component dialog box.
4. In the Choose Custom Component dialog box, locate the custom component file (with the .CC extension) that contains the component you want to put on the Component palette. Choose Open.
5. The path name to the selected custom component file now appears in the Set Up Custom Components dialog box.
6. With the desired .CC file selected, choose OK. The custom components you have saved in the .CC file appear on the Custom page of your Component palette (in both the Form and Report designers), ready to use just like any other component.

Removing custom components from the Component palette

To remove a custom component from the Component palette,

1. Choose File | Set Up Custom Components.
2. In the dialog box that appears, select the file that contains the custom component, and choose Delete.

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

Chapter 5 Accessing and linking tables

Chapter

5

Accessing and linking tables

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

This section discusses the following topics:

- The dBASE PLUS data model
- Linking a form or report to tables
- Creating master-detail relationships
- Creating and using a DataModule

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

Note

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

The dBASE data model

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

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

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

Query objects

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

Note

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

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

SQL property

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

```
select * from BIOLIFE
```

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

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

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

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

rowset property

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

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

Each Query object generates only one rowset, but you can add multiple Query objects to a form to use multiple rowsets from the same table, or from different tables. Using multiple Query objects also allows you to take advantage of dBASE PLUS's built-in master-detail linking.

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

Rowset objects

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

The row cursor and navigation

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

Synchronizing cursor movement in master-detail rowsets

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

Note

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

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

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

- *next*( ) moves the cursor a specified number of rows relative to its current position.
- *first*( ) goes to the first row in the rowset.
- *last*( ) moves to the last row.
- *goto*( ) uses the value returned by *bookmark*( ) to move back to that specific row.

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

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

Rowset modes

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

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

Rowset events

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

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

Row buffer

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

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

Field objects

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

value property

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

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

Important

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

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

If you leave out *value*,

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

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

Using *dataLink*s

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

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

Database objects

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

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

Accessing a database

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

Before you can use a Database object, you must set up BDE to access the database by using the BDE Administrator (available from the dBASE PLUS program group). .

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

Database-level security

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

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

Database-level methods

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

Default Database object

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

If you're accessing other table types, you need to use the Database object.

Session objects

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

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

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

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

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

StoredProc objects

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

The stored procedure can:

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

DataModRef objects

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

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

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

Note

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

Linking a form or report to tables

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

To see your tables listed in the Navigator:

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

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

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

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

Linking to a table automatically

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

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

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

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

Linking to a table manually

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

For tables you're accessing through a BDE alias,

1 Drag a Database object from the Component palette to the form or report design surface. (One Query object is added along with it.)

- Assign the BDE alias to the *databaseName* property.
- Set its *active* property to *true*.

2 For databases that require a login, you must either log in or set the Database object's *loginString* property, so that the table will open without requiring a password or ID to be entered. (Your login name and password must be set up by your database administrator.)

3 Select the Query object.

- Type the SQL query statement you want in the Query object's SQL property. Your SQL query can access any number of tables in the database. Some servers are case-sensitive for the table name; some may require quotation marks (Oracle, for example).

- Assign the Database object to the Query object's *database* property. This must be done before activating the query.
- Set the Query object's *active* property to *true*.

4 Add additional Query objects, if needed for other tables, and set their properties as in step 2. (If you want to drag a table from another database, be sure to first select the desired alias from the Navigator's Look In drop-down list, or in the case of BDE-standard tables without an alias, use the Navigator to locate the desired directory.)

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

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

Note

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

Procedure for using a Session object

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

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

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

Calling a stored procedure

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

To call a stored procedure,

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

Using local and remote tables together

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

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

Creating master-detail relationships (overview)

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

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

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

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

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

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

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

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

Using an SQL JOIN statement

In the Reports Designer use a single Query object whose SQL property contains an SQL JOIN statement linking the master table and the detail table. Both tables must be indexed on a common field.

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

By using two joined tables, you gain several advantages:

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

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

1. Add a Query object to the design surface.
2. Select the Query object. In the Inspector, click the wrench tool beside the SQL property to display the SQL Property Builder.
3. Do one of the following:

- Write an SQL JOIN query in the SQL Property Builder
- Locate a query you've already written

Note

Including image fields slows performance.

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

Linking master-detail in local tables

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

To link local master-detail tables,

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

Using the *masterSource* property

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

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

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

1. Drag the two tables onto the design surface of the designer you're working in.
2. Select the Query object of the detail table and set its *active* property to false.

3. Change the SQL statement in the Query object's SQL property to use host variables. For example, in a master-detail report on CUSTOMERS and ORDERS, you might use this:

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

assuming that ORDERNO is the exact field name in the table.

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

What is a DataModule?

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

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

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

Creating a DataModule

1. Choose File|New|Data Module, and choose Designer OR from the Navigator choose the Data Modules tab and double click on the [new Data Module] or [new Custom Data Module] icon. The Data Module Designer is similar to the Form designer, except that it has no grid and the Component palette contains only data access components.

2. Drag the components you need onto the design surface and set their properties (SQL and so on) and write their event handlers. Press F12 to toggle between the Source editor and the visual designer.

3. When everything is set up as you want it, save the data module (File|Save). It is saved with a .DMD extension.

Creating business rules in a dataModule

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

Using a DataModule

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

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

The properties, event handlers, and methods you set for components in a DataModule apply consistently to all forms and reports that use the module.

BDE, ADO, and ODBC… what is the difference?

Starting with dBASE PLUS 9, a new data access layer has been added to the product. The new layer has been around for some time, was developed by Microsoft and is called ADO or Active Data Objects.

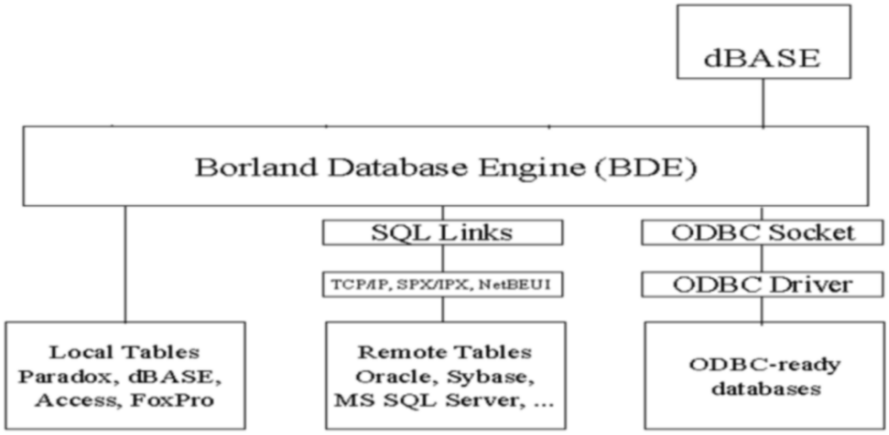
Ironically, ADO is much like the BDE in the way it tackles cross-database connections. It uses a Client API layer and also a middle layer to communicate through a driver to the exact database on the backend.

The next question for many developers and users of dBASE is which one do I use? That depends on the current state of the application you are trying to deploy.

This paper will outline the architectures for BDE (Borland Database Engine), ADO (Active Data Objects), cover ODBC (Open DataBase Connectivity), and finally finish with a discussion and flowchart on which is approach is best for the users of dBASE.

What is the BDE?

In the early to mid-90's, Borland had an abundance of database and data-access technologies. However, they wanted to come up with a one engine fits all, which was really the precursor to the ODBC standard. The BDE or Borland Database Engine as it is known, is just that product. It has native interfaces to both dBASE and Paradox files, has an ODBC Socket included, and also another "enterprise" socket used by the SQL Links product. Below is the architecture diagram that sums up the technology.



High level architecture of BDE

This was an incredibly rich interface to communicating with many underlying databases regardless of the method used, it was fast, full of functionality, and allowed for direct native access to dBASE (.DBF) and Paradox files (.DB). The BDE code is still owned by Embarcadero, the owners of the Borland codebase. In other words, dBase, LLC., does not own and cannot make changes to the BDE in any way. This means when there are bugs, limitations, or wanted features, dBase the company is unable to make changes to that technology.

The biggest problem with the BDE is that investment and development stopped back in 2001 when Borland EOL (End Of Life) the product. That being said, the BDE is still used by millions around the world and will be used into the future. There are, however some limitations that cannot be overcome at this time, the biggest being that it is 32-bit compiled, which means that the product will not be natively compiled on 64-bit machines. Therefore, sometime in the future, the BDE will not work on those machines if they do not support 32/64 bit as is found today.

So if you are using the BDE and are not having issues or have worked around a vast majority of the issues and have a working product, then it would most likely be suggested that unless you need to change databases, or have some other external requirement, the BDE is the best choice in that situation. Other situations may not have the same outcome and will be covered in the last part of the chapter.

What is ADO?

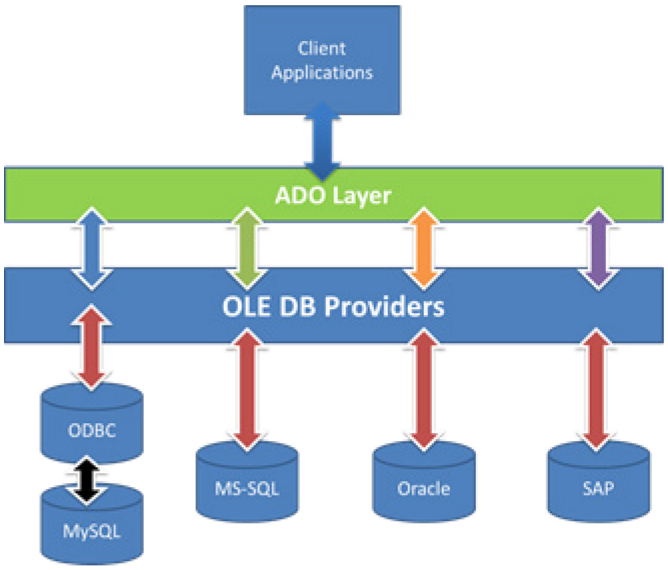
ADO, which stands for Active Data Objects was created by Microsoft to be the solution against ODBC originally. While ADO has a wide audience, most of the ADO thunder was taken by ADO.Net in the past few years.

Just to be clear, the ADO included in dBASE PLUS 9with ADO is the ADO based on OLE DB and NOT ADO.NET.

As Microsoft defines ADO:

> "Microsoft ActiveX Data Objects (ADO) enable your client applications to access and manipulate data from a variety of sources through an OLE DB provider. Its primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. ADO supports key features for building client/server and Web-based applications." Microsoft – MSDN

As you can see by the below architecture, it is very similar to the BDE.



High level architecture of ADO

The advantages of ADO currently consist of Microsoft supported technology, which means the ADO-layer is up to date with the latest Microsoft Operating Systems. It also means it behaves much better on 64-bit machines and of course has full 32-bit support.

ADO, much like the BDE, also supports the concept of an ODBC Socket, which allows ADO to use standard ODBC drivers that use the Socket as a conduit through the ADO layers. This means that connecting to additional databases with a standard ODBC driver is available to the technology as well.

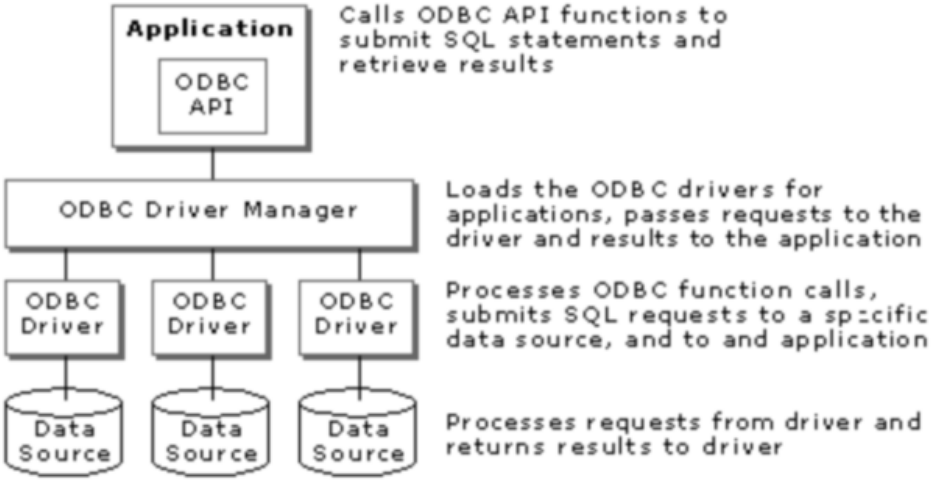
ADO also has up-to-date network support and multi-user support built in. This gives an advantage to ADO over the BDE in most "outside" databases, meaning not .dbf and .db files respectfully.

The one drawback to the ADO approach is that it does not support native dBASE (.dbf) or Paradox (.db) files out of the box, which means you will have to use a driver for that connection. In this case, most ODBC drivers are still based on Level-5 compatibility and not the Level-7 that the BDE is used to communicating ultimately, this means that ADO is a poor choice if you plan on connecting and using native dBASE or Paradox files.

What is ODBC?

The reason why we cover ODBC is the fact that it is the only way to communicate to additional databases using BDE and ADO. Yes the BDE has SQL Links, but they are fairly old and things have changed in the database market since they were introduced. The same can be said about ADO, there are database specific drivers made using the OLE DB interface, however many now bypass that approach and use the built-in ODBC Socket in ADO for that type of interaction between data and databases across a network.

The diagram below shows the layers a common ODBC approach uses:



Taken from Microsoft / MSDN

One of the interesting facts about ODBC is the following, taken from Wikipedia:

> ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS. The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS. An ODBC driver can be thought of as analogous to a printer or other driver, providing a standard set of functions for the application to use, and implementing DBMS-specific functionality. An application that can use ODBC is referred to as "ODBC-compliant." Any ODBC-compliant application can access any DBMS for which a driver is installed. Drivers exist for all major DBMSs and even for text or CSV files. [Taken from Wikipedia]

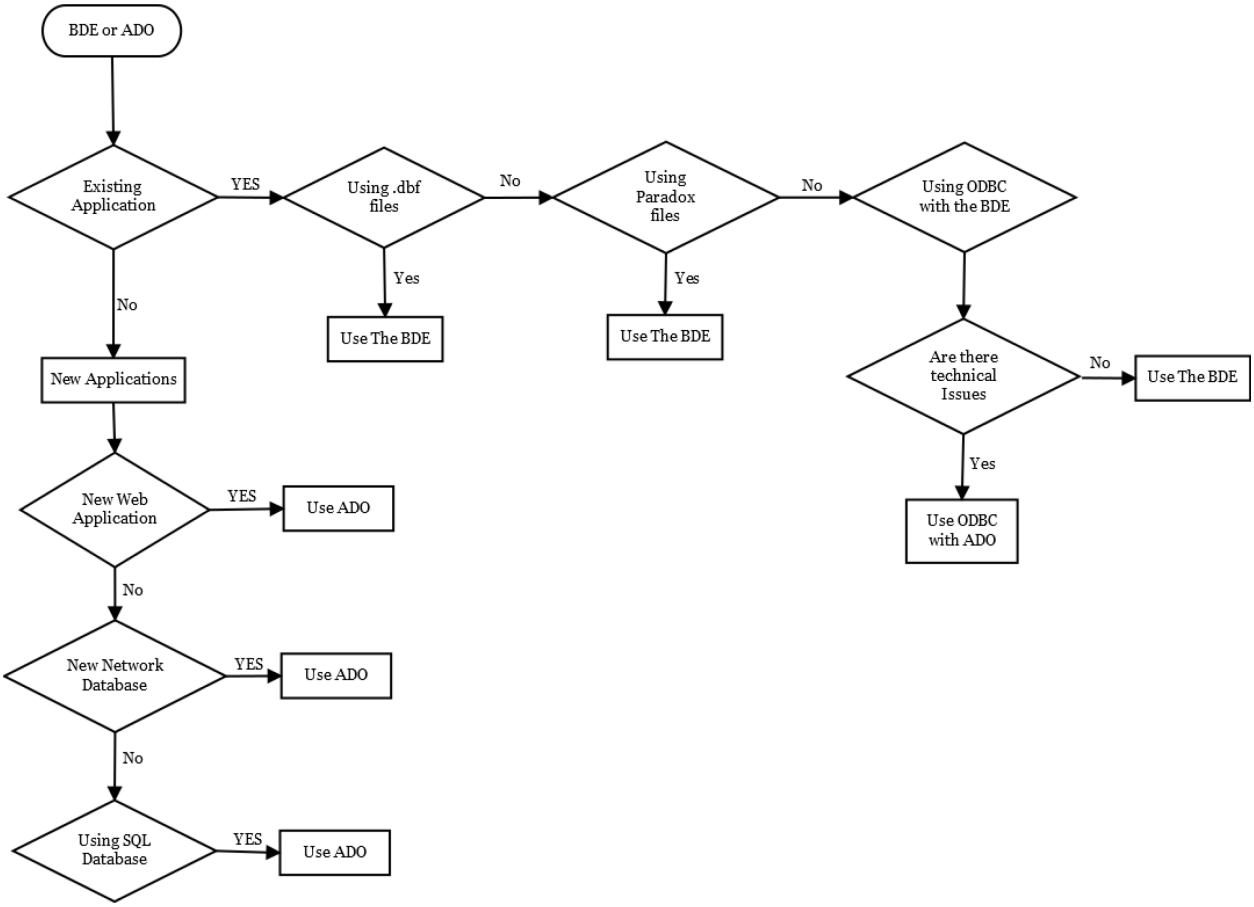
This means that using either the BDE or ADO, the dBASE developer has a wide variety of databases that can be connected to and manipulated.

So really, it is between BDE and ADO?

CORRECT! Since dBASE does not currently support a native ODBC socket at this time, the only way to get cross-database connectivity is to use either BDE or ADO. ODBC needs to have the Socket in order to communicate.

Existing customer, how to choose?

The following is a simple flowchart to help dBASE developers ascertain which technology is best for which solution.



High level questions for picking a data-access solution

The flowchart is fairly straight forward: if you have an application using the BDE and it is working, stay with the BDE. If you are using the BDE and having issues, it may be worth investigating moving to ADO. If you were looking to start a new project, then the recommendation would be to use ADO for future development. The main reason again for this recommendation is that dBASE cannot guarantee when, or what, operating system from Microsoft will not allow it to run. ADO is currently being supported and will be for the foreseeable future.

This flowchart is meant as a helper and not an end-all, be-all, for picking a technology. Each technology decision must be made with the best information available at the time. The BDE may be the perfect choice in some cases of new development and ADO would not be the best choice.

Chapter 6 Using FormReport Designers

Chapter

6

Using the Form and Report designers

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

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

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

Note

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

The designer windows

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

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

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

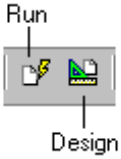
By default, a grid appears when you start the Form and Report designers, and objects are constrained to line up along the grids (Snap To Grid). In addition, vertical and horizontal rulers appear.

Both designers have the following tools:

- Component palette for dragging user-interface elements and data-access objects to the design surface
- Field palette for dragging linked fields to the design surface
- Inspector for setting properties and writing event handlers and other methods
- Format toolbar for formatting Text objects
- Alignment toolbar for aligning objects
- Layout, Format, and Method menus
- Status bar to show you your location on the design surface, show you what object is selected, and to give you instructions and other information

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

Design and Run modes



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

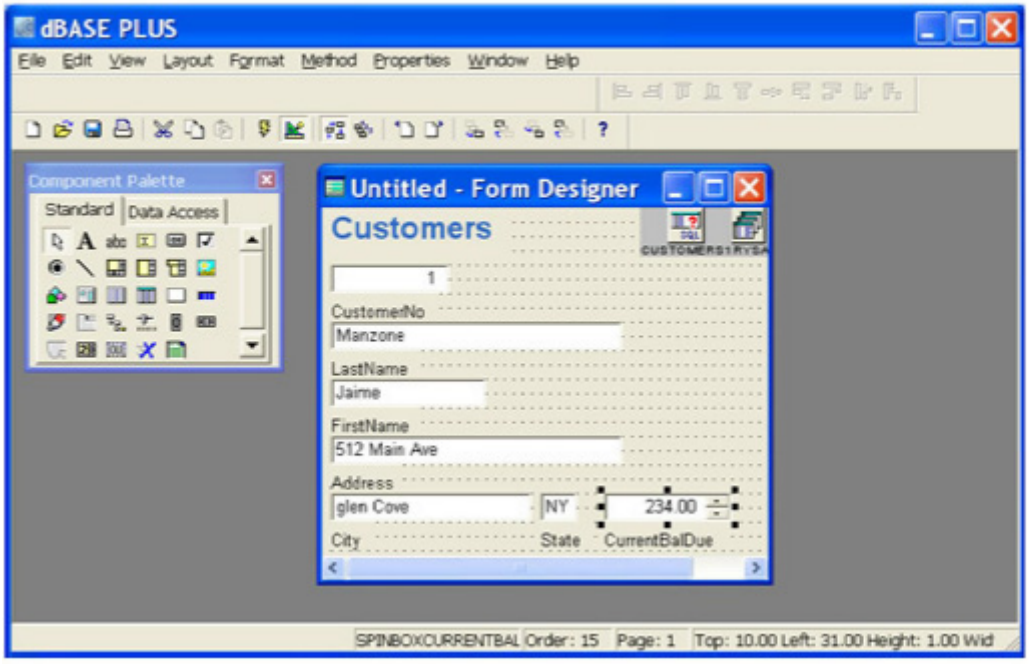
- Use Design mode to design the appearance and behavior of the form or report and the components you put on it.
- Use Run mode to see how a form or report looks when running. In Run mode, the components become active. For example, you can enter data into an entryField control and edit data that's already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

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

The Form Design Window

A form in the Form Designer appears on the desktop a shown in the following figure.

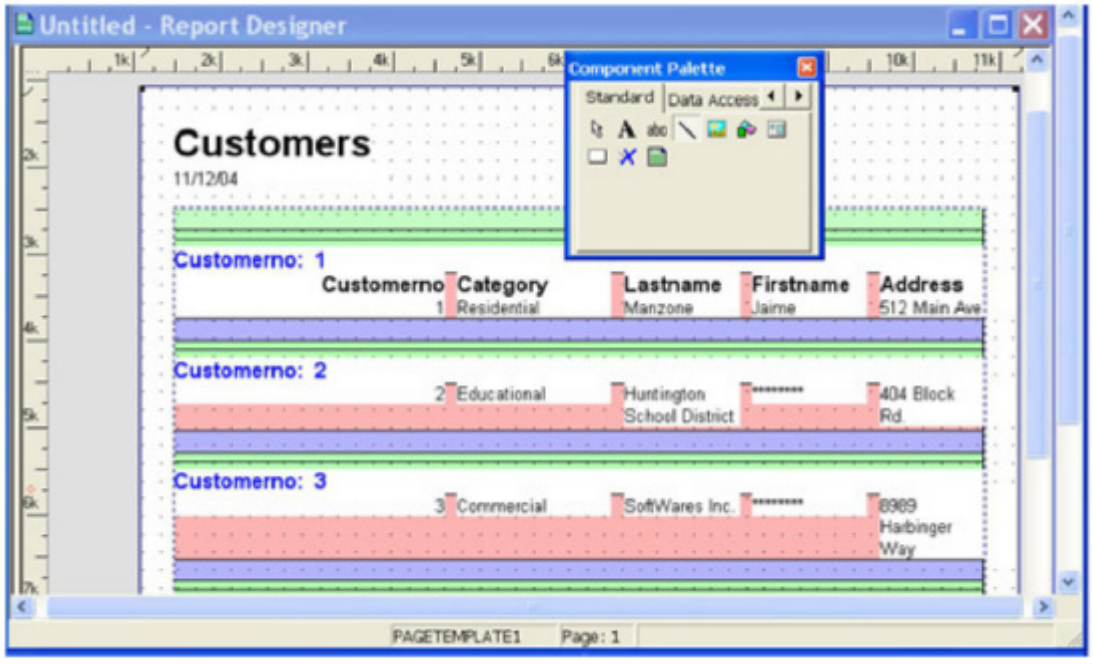
Figure 0.1 Form Designer with a wizard-created form



The Report Design window

A report in the Report designer appears on the desktop as shown in Figure 6.2

Figure 0.1 Report Designer with a wizard-created report



The report design surface has several objects the form design window does not, for example, pagetemplate and streamframe. These objects are necessary for formatting report pages. See Chapter 11, “Designing reports”, for information on working in the report design window.

The visual design is reflected in your code

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

Component palette

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

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

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

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

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

Standard page

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

Table 6.1 Standard controls

Component	Use to . . .	Example/Explanation
Text 	Display text that cannot be edited by users. The text can be any alphanumeric characters allowed in a character expression	Use for a field label, heading, instruction, prompt, or any other non-editable display text. Format with the Format menu or Format toolbar.
TextLabel 	Display information on a form or report, wherever features such as word-wrap and HTML formatting are not required.	TextLabel is a simple, light-duty object which consumes fewer system resources than the Text component. The TextLabel component does not support in-place editing on design surfaces. The *text* property of the TextLabel component may contain character string data only.
EntryField	Let a user enter a single value, text or numbers, into a data-entry field	Example: Data entry area for entering a value for a particular field of a table. Must be *DataLink*ed to the table field.

		
PushButton 	Let a user perform a task with a single mouse-click.	A control that a user can click to execute code that you attach. (Sometimes called a command button.)
CheckBox 	Let a user toggle between two choices of a logical value. Or choose a number of options that are not mutually exclusive.	Check boxes often are arranged in groups to present choices or options a user can turn on or off. Any number of check boxes in a group can be checked at a time
RadioButton 	Let a user select one choice among a group of mutually-exclusive possible values.	Example: A group of buttons labeled Credit, Cash, Check, Visa, and MC to choose among for entering only one of those values in a PAY_METHOD field of a table.
Line 	Organize elements visually	The line is a divider that may be extended vertically, horizontally, or diagonally to visually divide a form into sections. Users cannot edit or manipulate it.
Editor 	Display the contents of a text file or memo field.	Text exceeding the size of the box causes a scrollbar to appear. You can choose to allow users to edit this text.
ListBox 	Display values in a fixed-size, scrollable list box, from which a user can select one or more items.	The values in a list can be file names, records, array elements, table names, or field names.
ComboBox 	Combine an entry field and a drop-down list box. A user clicks the down-arrow button to display the list.	A combo box accepts a value typed into the entry field or selected from the drop-down list box.
Image 	Display an image.	Display area for a bitmap image stored in a binary field, resource file, or graphic file.
Shape 	Visually divide a form into sections, for example, to place related RadioButtons within a box.	A visual appearance element. By setting the component's *shapeStyle* property you can create rounded rectangles, rectangles, ellipses, circles, rounded squares, and squares. You can also set the line style, weight, and interior color.
Container 	Create moveable panels that can contain other components on a form.	Example: Moveable toolbars and palettes.
Browse 	Display multiple records in row-and-column format.	The Browse component is maintained for compatibility and is suitable for viewing and editing tables open in work areas. For forms that use Data Access use a Grid object instead.
Grid 	Display live table data in row/column format in a programmable component.	The Grid object is a multi-column grid control for displaying the contents of a rowset. The *dataLink* property is set to the rowset. Columns are automatically created for each field in the rowset

Rectangle 	Organize elements visually into boxes or create custom buttons.	A graphic element for boxing objects. You can set the size, line weight, and fill of the box. It can respond to mouse clicks and other events.
Progress 	Provide visual feedback to the user about the progress of long operations or background processes.	A rectangular bar that "fills" from left to right, like that shown when you copy files in the Windows Explorer. Use *position* to set a default position for the progress bar. At run time, *position* tracks the exact location as values increment. Use *max* and *min* to set the range of *position*. By default, the progress meter advances by a value of one.
PaintBox 	Create custom form controls	The PaintBox provides a window space in which you can call API functions in Windows. Users never interact with it directly. dBASE PLUS does not paint the area.
NoteBook 	Make a multipage dialog box, with labeled tabs to display sections of information or groups of controls within the same window. See TabBox for full-size tabbed forms.	You might use the Notebook control to create a tabbed dialog box with different groups of controls on each tabbed page. The Desktop Properties dialog box is a good example of this. Use the DataSource Property Builder to name or add tabbed pages to the window. Then drag the components you want to each tabbed page.
TreeView 	Display and control a set of objects as an indented outline based on their logical hierarchical relationships. The control includes buttons that allow the outline to be expanded and collapsed.	Use a tree view component to display the relationship between a set of containers or other hierarchical elements. You might use the TreeView as a way to select items from nested lists, much like the hierarchical view in the left pane of the Windows Explorer.
Slider 	Define the extent or range of values. By moving the slider along the trackbar, the user can change the current value for the control	You can set the trackbar orientation as vertical or horizontal, define the length and height of the slide indicator and the slide bar component, define the increments of the trackbar, and whether to display tick marks for the control. Examples: A volume control to play back sound files, or a color saturation adjuster for an image viewer
Vertical scroll bar 	Allow users to vertically scroll a grouping of controls, or a large control that has no integrated scroll bars	Example: A custom dialog box containing an area filled with many file icons.
Horizontal scroll Bar 	Allow users to horizontally scroll a grouping of controls, or a large control that has no integrated scroll bars	Example: A custom dialog box containing an area filled with many file icons.
TabBox 	Group related data items on overlapping pages with labeled tabs	Use a TabBox to display multiple pages the full size of the form. A user selects a tab to display the items on the TabBox. Similar to Notebook, except for the full form size.
SpinBox 	Provide up and down arrows to assist changing a numeric value.	You can type a number into the numeric entry field or can increment or decrement the number by clicking the up and down arrows.
OLE 	Create an object linking and embedding (OLE) client area in a form, in which you can embed, or link to, a document from another application	Using an OLE control, a document from another application, for example, a sound file from a sound recorder application, can be opened from your dBASE PLUS application

ReportViewer 	Displays a report in a sizeable frame	The report is executed when the form is opened.

Data Access page

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

Note

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

This table describes the data objects available from the Data Access page of the Component palette. For details on the dBASE PLUS Data Model and use of the Data objects, see Chapter 5, “Accessing and linking tables”.

Table 6.2 Data Access

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

Data Buttons page (forms)

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

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

Table 6.3 Shading Properties in the Table Designer

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

Report page

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

Table 6.4 Components specific to reports

Component	What it is	What it does

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

Custom page

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

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

Using ActiveX (*.OCX) controls

To use an ActiveX control in your forms and reports,

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

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

2 Select the desired controls.

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

The Field palette

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

To open the Field palette, either

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

Figure 0.1 Field Palette

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

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

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

If no active Query object exists on the form or report, the Field palette is empty, showing only the Pointer button. When you begin to design a data-aware form or a report, first add a Query object and set its *sql* property to the appropriate SQL statement and its *active* property to *true*. If you drag a table from the Navigator to the design surface, this automatically creates a Query object that selects all the records in that table and links the table to the form or report. See Chapter 5, “Accessing and linking tables”, for more details.

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

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

The Inspector

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

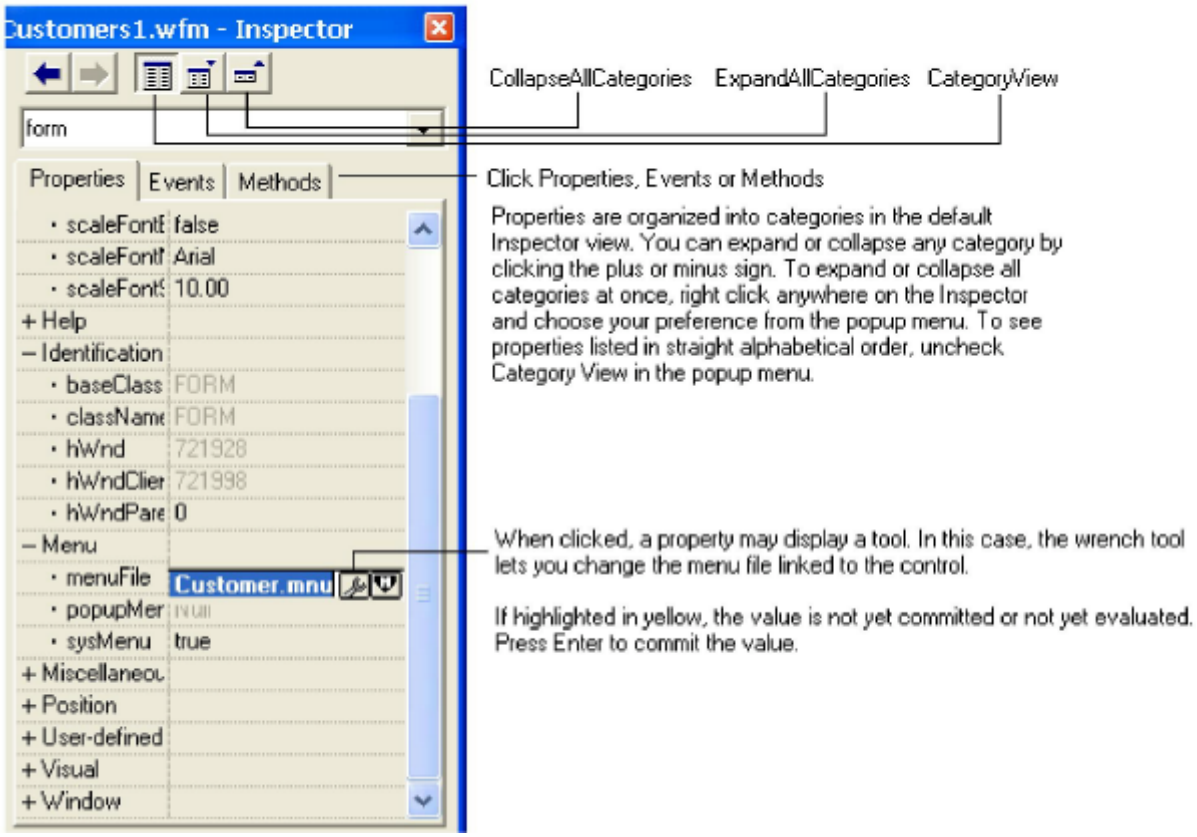
- Press F11.
- Choose View | Inspector (this command is a toggle).
- Right-click the selected component and choose Tool Windows | Inspector from the context menu.

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

Note

You cannot change methods for multiple selections.

Figure 0.1 Using the Inspector



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

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

Properties page of the Inspector

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

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

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

  Note

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

- Press Ctrl+Enter in the value column to rotate through a list of properties or to toggle logical values, or double-click the value column to do the same.
- Select a value from a history list or other drop-down list, when available.
- Click the wrench tool button that appears to the right of the property value. Tools are not available for every property.

  
  The tool button may produce

  - A property builder in which you can build or select a value. For example, you can display the

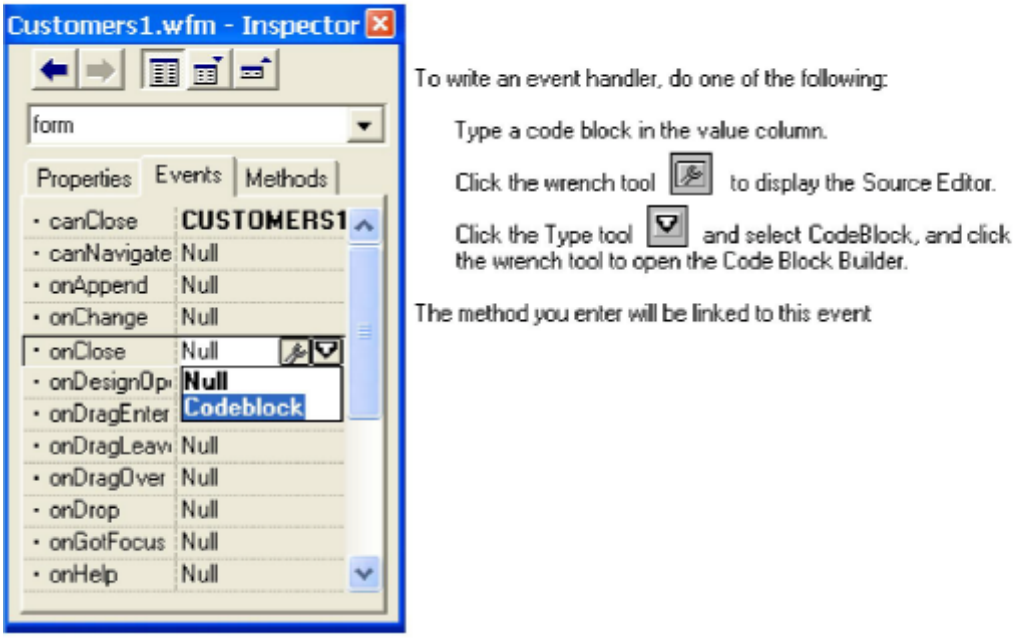
Color property builder to set the color for an object.

- The String Builder dialog box, which makes it more convenient to type a long string.

Events page of the Inspector

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

Figure 0.1 Events page of The Inspector



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

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

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

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

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

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

You can also link and unlink events by using the Method menu from within the Source editor. See "The Method menu"

Methods page of the Inspector

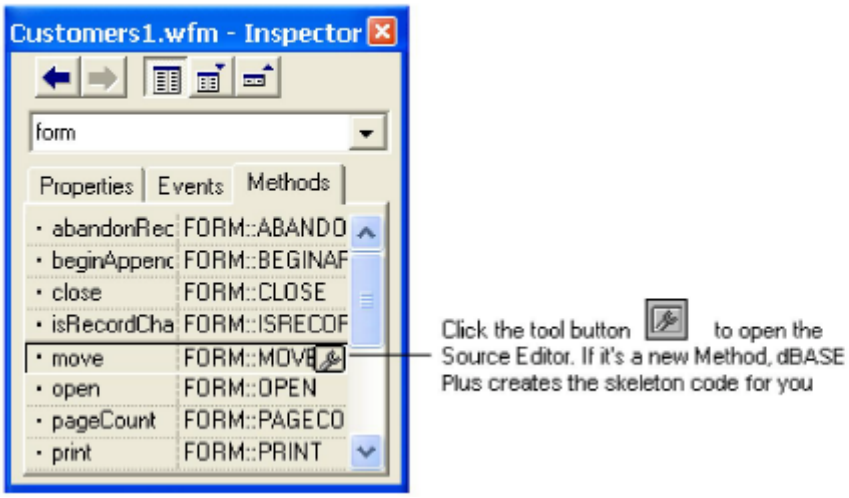
The Inspector's Methods page displays the current object's built-in methods, that is, the methods pre-defined for the component. You can call these methods with methods you create in the Source editor. Methods you create in the Source editor can be inspected on the Methods page.

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

Note

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

Figure 0.1 Methods page of The Inspector



The Method menu

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

Table 6.5 Method menu commands

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

Manipulating components

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

Placing components on a form or report

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

Note

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

To place a component,

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

Note

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

Alternatively, you can add a component in these ways:

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

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

Special case: container components

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

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

Note

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

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

Selecting components

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

To select a component, do one of the following:

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

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

Note

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

Moving components

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

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

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

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

Cutting, copying, pasting, deleting components

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

Undoing and redoing in the designers

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

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

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

Aligning components

You can align components by using the Layout | Align menu commands or the corresponding toolbar buttons (to display the Alignment toolbar, choose View | Tool Windows, and check Alignment Toolbar.) These commands adjust the position of objects in relation to each other or in relation to the form or report. To find out what each option does, highlight it and read the explanation in the status barer press F1. See Figure 6.7 for a summary.

Resizing components

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

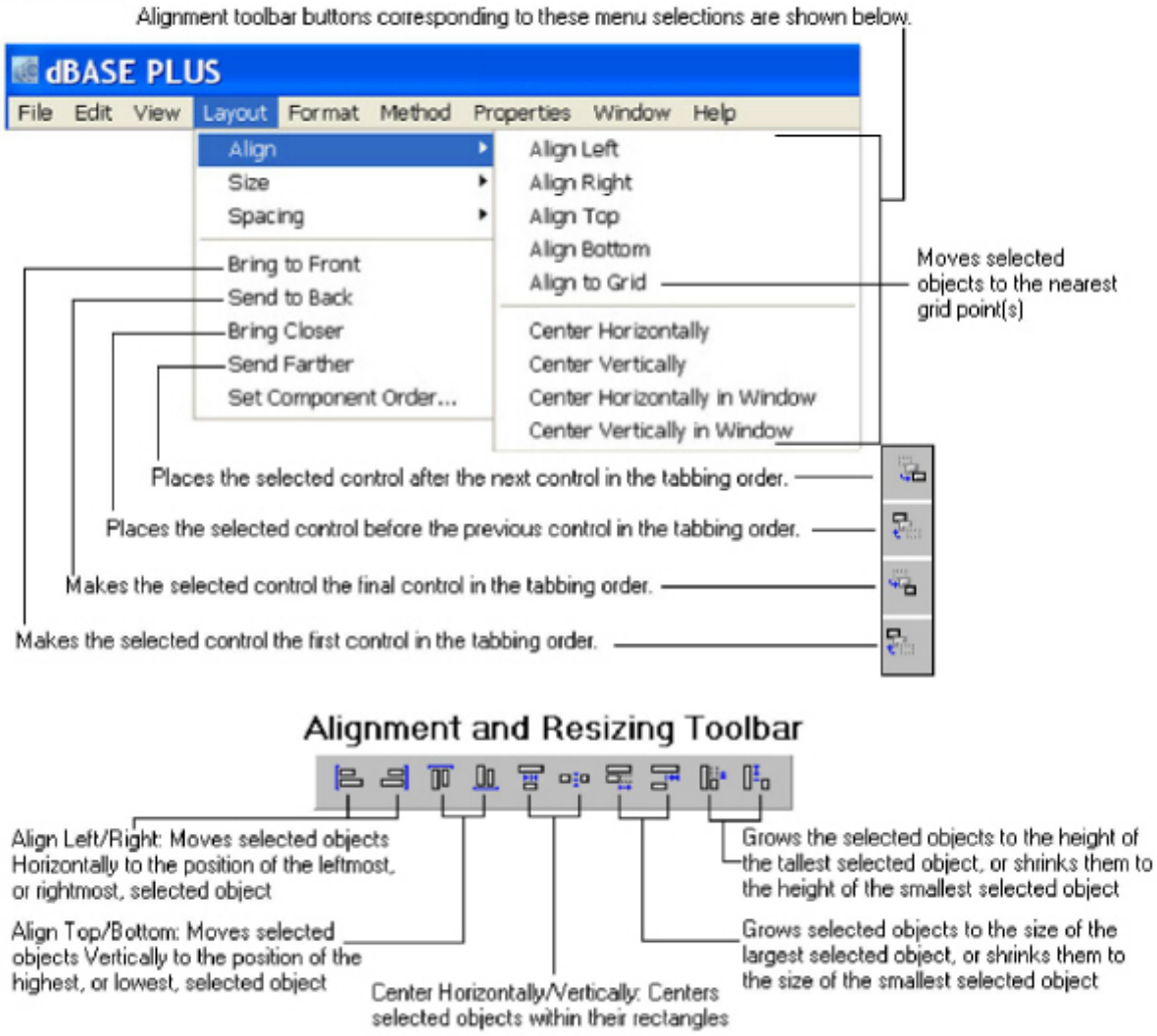
- Place the mouse pointer on one of its handles. When the pointer turns into a double-headed arrow, drag the handle to size the component the way you want.
- Press Shift+any arrow key to resize it in the direction of the arrow.

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

To change the sizes of multiple objects to conform to one size, choose an option from the Layout | Size menu or the corresponding button in the Alignment toolbar. (To display the Alignment toolbar, choose View | Tool Windows, and check Alignment Toolbar.) To find out what each option does, highlight it and read the explanation in the status bar

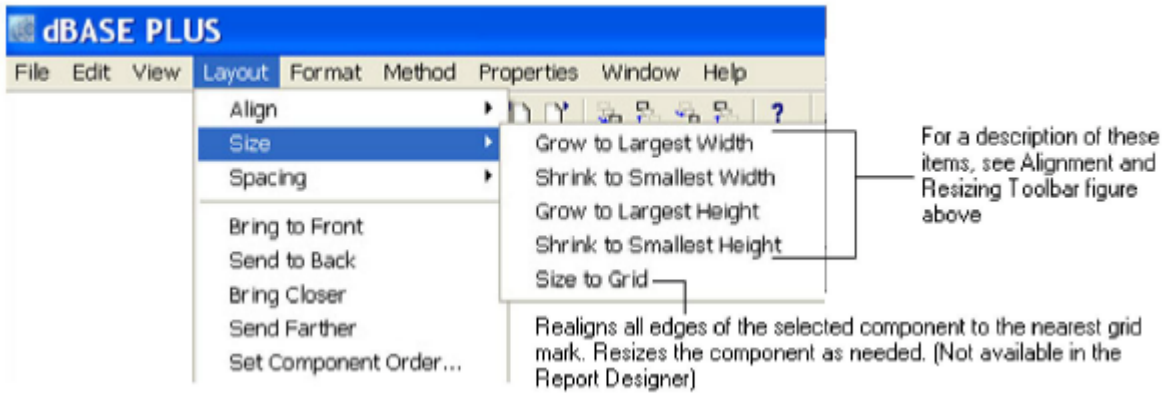
:

Figure 0.1L



.

Figure 0.1 Layout | Size commands



Spacing components

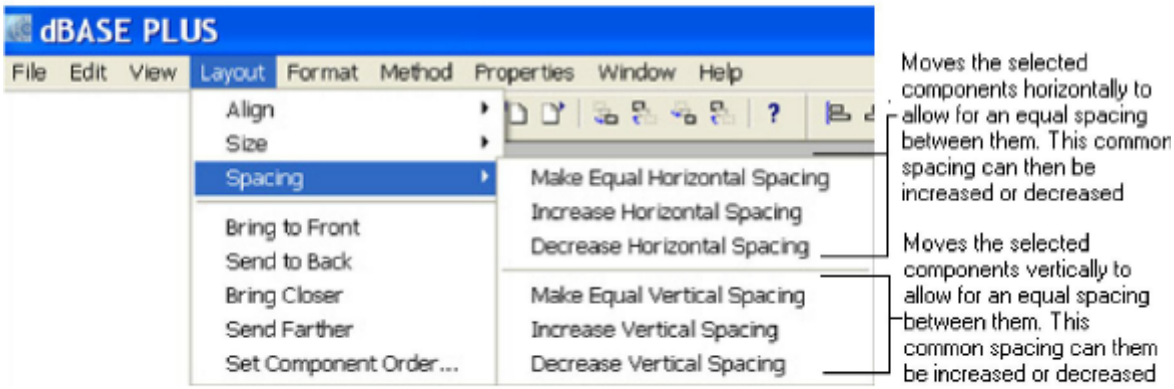
To change the spacing of components in the Form designer, select the components, and choose an option from the Layout | Spacing menu.

Note

The Spacing menu is not available in the Report designer.

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

Figure 0.1 Layout | Spacing commands

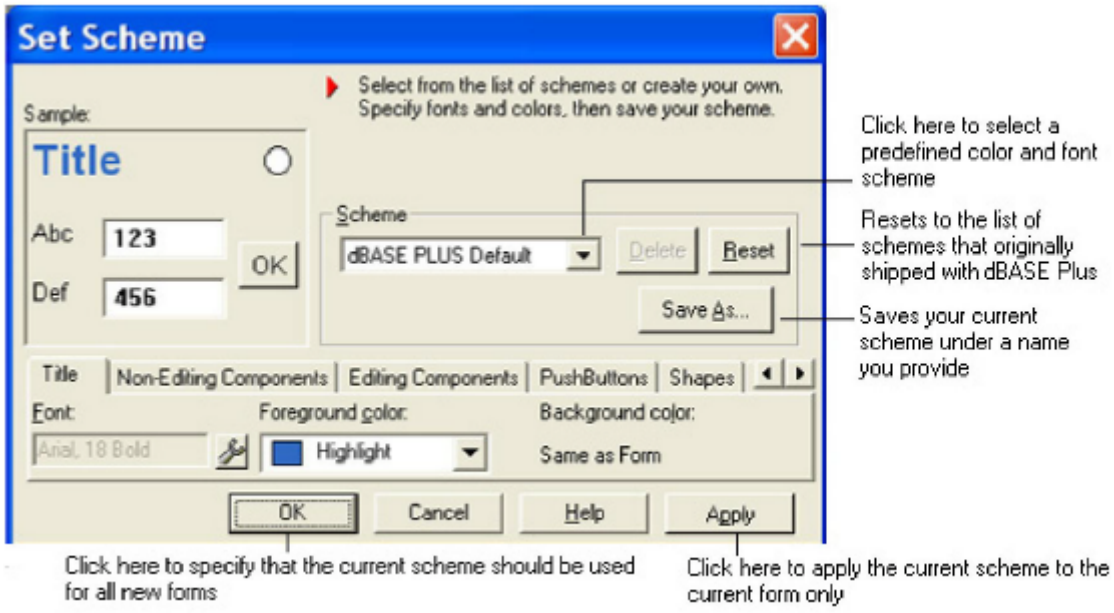


Setting a scheme (Form designer)

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

- Choose a predefined scheme
- Set your own scheme

Figure 0.1 Set Scheme dialog box



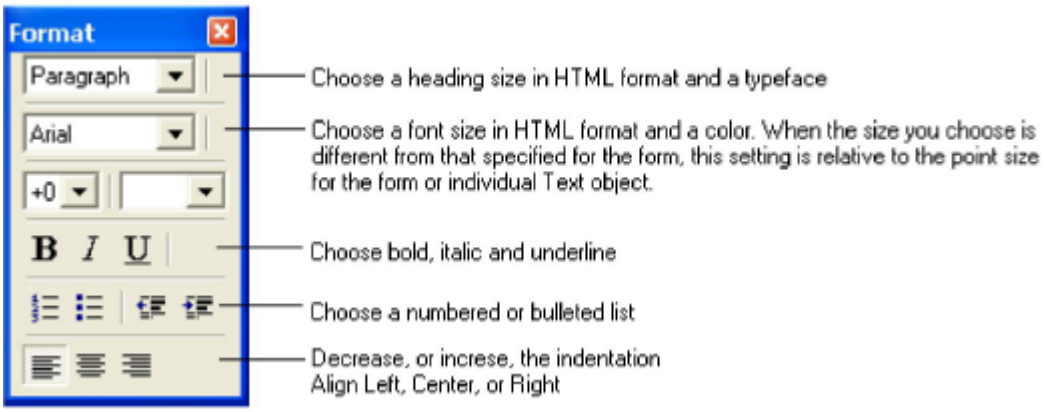
Editing a Text object

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

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

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

Figure 0.1 Format toolbar



Note

Once you have used the Format toolbar or menu to change parts of a Text object, you cannot use the Inspector to override what you have done. Use the Format toolbar or menu, instead. However, the Inspector *will* make changes you specify to anything you haven't already changed by using the Format toolbar or Format menu.

Saving, running, and printing forms and reports

To save a form or report design, either

- Click the Save toolbar button.
- Choose File | Save or File | Save As.

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

Opening a form or report in Run mode

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

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

Printing a form or report

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

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

Chapter 7 Create menus toolbars

Chapter

7

Creating menus and toolbars

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

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

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

Attaching pulldown menus to forms

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

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

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

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

Attaching popup menus to forms

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

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

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

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

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

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

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

Creating toolbars and attaching them to forms

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

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

Creating a reusable toolbar

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

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

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

Note

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

Attaching a reusable toolbar

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

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

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

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

Of course, you also need to provide a way to restore the toolbar if the user has closed it. You can do that by also adding the integration code (or codeblock) to the *onClick* event of another control, such as a menu item or button. Should the toolbar already be running when it is summoned, *findInstance*(_) will let you know and let you block the creation of a new instance.

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

Creating a custom toolbar

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

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

```
** END HEADER -- do not remove this line
*
* Generated on 08/20/97
*
parameter bModal
local f
f = new tooltestForm( )
if (bModal)
  f.mdi = .F. // ensure not MDI
  f.ReadModal( )
else
  f.Open( )
endif

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

this PUSHBUTTON = new PUSHBUTTON1(this)
with (this.PUSHBUTTON1)
  onClick = class::show_toolbar
```

```
  height = 1.1176
  left = 4
  top = 2
  width = 15.875
  text = "PUSHBUTTON1"
  metric = 0
  fontBold = false
  group = true
endwith

// {Linked Method} Form.onOpen
function Form_onOpen

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

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

ENDCLASS

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

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

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

Creating menus with the designers

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

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

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

The designer menu

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

You can use these shortcuts to insert an item before the current item (Insert Menu Item, Alt+MN or Ctrl+N), start a new submenu (Insert Menu, Alt+MM or Ctrl +M) or insert a separator (Alt+MT or Ctrl+T) before the current item in a pulldown or submenu. You can also delete the current item with Alt+MD or Ctrl+U (also see, "Adding, editing and navigating", on page 118.

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

The last item on the Menu list—Toggle Type (Alt+MO)—is available for use on those occasions when you change your mind about the type of menu you want. It automatically switches the currently selected designer and converts its contents from pulldown style to popup style—or vice versa—any time.

Building blocks

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

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

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

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

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

Adding, editing and navigating

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

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

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

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

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

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

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

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

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

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

Features demonstration

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

1. Open a new pulldown Menu designer window.
2. Type &File (including the ampersand) at the cursor, then press Enter. Type &Form into the new item entry box, then press Tab. A new item entry box appears to the right of the current entry. Type &Close into this box.
3. If it isn't already open, press F11 to open the Inspector. Choose the Events tab, then type form.close( ) into your Close item's onClick event.
4. Back at the Menu designer, select the top-level "&File" item.
5. Press Tab. A new top-level item entry box appears. Press Alt+NE to insert a complete menu of basic editing commands. Now press the Tab key again for one more top-level item entry box.
6. Press Alt+NW. This time, a new top-level "&Window" item is created. This item has no subentries yet, but it will later.
7. Save the menu as MTEST.MNU, then close the Menu designer.
8. Open a new form in the Form designer. Add an entryfield control to the form.
9. Click on the form background. If it's not already in view, press F11 to view the Inspector. Click the tool button on the form's *menuFile* property (Menu category), choose your MTEST.MNU file, and click OK. Keep other form properties at their default settings.
10. Press F2 to save (MTEST.FRM, for example) and run the form.

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

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

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

Note

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

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

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

Examining menu file code

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

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

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

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

```
** END HEADER -- do not remove this line
//
// Generated on 10/24/97
//
parameter formObj
new mtestMENU(formObj, "root")

class mtestMENU(formObj, name) of MENUBAR(formObj, name)

  this.MENU2 = new MENU(this)
  with (this.MENU2)
    text = "&File"
  endwith

  this.MENU2.MENU3 = new MENU(this.MENU2)
  with (this.MENU2.MENU3)
    text = "&Form"
  endwith

  this.MENU2.MENU3.MENU7 = new MENU(this.MENU2.MENU3)
  with (this.MENU2.MENU3.MENU7)
    onClick = {;form.close( )}
    text = "&Close"
  endwith

  this.MENU12 = new MENU(this)
  with (this.MENU12)
    text = "&Edit"
  endwith

  this.MENU12.UNDO = new MENU(this.MENU12)
  with (this.MENU12.UNDO)
    text = "&Undo"
    shortCut = "Ctrl+Z"
  endwith

  this.MENU12.CUT = new MENU(this.MENU12)
  with (this.MENU12.CUT)
    text = "Cu&t"
    shortCut = "Ctrl+X"
  endwith

  this.MENU12.COPY = new MENU(this.MENU12)
  with (this.MENU12.COPY)
    text = "&Copy"
    shortCut = "Ctrl+C"
  endwith

  this.MENU12.PASTE = new MENU(this.MENU12)
  with (this.MENU12.PASTE)
    text = "&Paste"
```

```
    shortCut = "Ctrl+V"
  endwith

  this.MENU17 = new MENU(this)
  with (this.MENU17)
    text = "&Window"
  endwith

  this.MENU11 = new MENU(this)
  with (this.MENU11)
    text = ""
  endwith

  this.windowMenu = this.menu17
  this.editCutMenu = this.menu12.cut
  this.editCopyMenu = this.menu12.copy
  this.editPasteMenu = this.menu12.paste
  this.editUndoMenu = this.menu12.undo

endclass
```

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

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

The remaining Editmenu properties function in a similar fashion.

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

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

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

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

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

And then create a method to handle the event:

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

Changing menu properties on the fly

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

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

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

Menu and menu item properties, events and methods

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

Note

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

Table 7.1 Menubar and popup root properties, events and methods

Property	Description
alignment (popup only)	Lets you align items on your popup menus and submenus. Options are left-aligned, centered, and right-aligned. Default is left-aligned.
baseClassName	Identifies the object as an instance of the Menu, MenuBar or Popup class.
className	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
editCopyMenu, editCutMenu, editPasteMenu, editUndoMenu (menubar only)	These four built-in objects are available for assignment to items on a preset Edit menu on a pulldown menu (menubar class). To access properties for these objects, click the object's Tool button.
left (popup only)	Sets the position of the left border of the popup. Default is 0.00.
name	String used to reference the root menu object. Except for Edit and Window menu names (which use the defaults EDIT and WINDOW), default for custom menus is ROOT. A reference to a default item would thus be this.root.menuNN, where NN is a system-assigned item number.
top (popup only)	Sets the position of the top border of the popup. Default is 0.00.
trackRight (popup only)	Logical value (default true). Determines whether popup menu items can be selected with a right mouse click. If set to false, the popup menu is still opened with a right-click, but items must be selected with a left-click.
windowMenu (menubar only)	This built-in object is available for assignment to items on a preset Window menu on a pulldown menu (menubar class). To access properties for the WindowMenu object, click the object's Tool button.
Event	**Description**
onInitMenu	Codeblock or reference to code that executes when the menu is initialized (when its parent form is opened).
Method	**Description**
open( ) (popup only)	Opens the popup menu.
release( )	Removes the menu object definition from memory.

Table 7.2 Item properties, events and methods

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

onHelp	Optional code that executes when the user presses F1. Use this to provide user information as an alternative to using the *HelpFile* and *HelpId* properties to define an online Help topic.

Method	Description
release( )	Removes the object definition from memory.

Toolbar and toolbutton properties, events and methods

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

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

Tip

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

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

Table 7.3 Toolbar properties, events and methods

Property	Description
baseClassName	Identifies the object as an instance of the ToolBar class
className	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
flat	Logical value (default true) which toggles the appearance of buttons on the toolbar from always raised (false) to only raised when the pointer is over a button (true).
floating	Logical value (default false) that lets you specify your toolbar as docked (false) or floating (true).
form	Returns the object reference of the form to which the toolbar is attached.
hWnd	Returns the toolbar's handle.
imageHeight	Adjusts the default height for all buttons on the toolbar. Since all buttons must have the same height, if *imageHeight* is set to 0, all buttons will match the height of the tallest button. If *ImageHeight* is set to a non-zero positive number, images assigned to buttons are either padded (by adding to the button frame) or truncated (by removing pixels from the center of the image or by clipping the edge of the image).
imageWidth	Specifies the width, in pixels, for all buttons on the toolbar.
left	Specifies the distance from the left side of the screen to the edge of a floating toolbar.
text	String that appears in the title bar of a floating toolBar.
top	Specifies the distance from the top of the screen to the top of a floating toolbar.
visible	Logical property that lets you hide or reveal the toolbar. Default is *true*.

Event	Description
onUpdate	Fires when the application containing the toolbar is idle, intended for simple routines that enable, disable or otherwise update the toolbar. Because this event fires continuously when the application is idle, you should avoid coding elaborate, time-consuming routines in this event.
Method	**Description**
attach( )	Attach (<form object reference>) establishes communication between the toolbar and the specified form and sets the Form property of the toolbar. Note that a toolbar can be attached to multiple MDI forms or to a single SDI form. For examples, see Help (search for "class ToolBar").
detach( )	Detach (<form object reference>) ends communication between the toolbar and the specified form, and closes the toolbar if it is not attached to any other open form.

Table 7.4 Toolbutton properties, events and methods

Property	Description
baseClassName	Identifies the object as an instance of the ToolButton class
bitmap	Graphic file (any supported format) or resource reference that contains one or more images that are to appear on the button.
bitmapOffset	Specifies the distance, in pixels, from the left of the specified Bitmap to the point at which your button graphic begins. This property is only needed when you specify a Bitmap that contain a series of images arranged from left to right. Use with *BitmapWidth* to specify how many pixels to display from the multiple-image Bitmap. Default is 0 (first item in a multiple-image Bitmap).
bitmapWidth	Specifies the number of pixels from the specified Bitmap that you want to display on your button. This property is only needed when you specify a Bitmap that contain a series of images arranged from left to right. Use with *BitmapOffset*, which specifies the starting point of the image you want to display.
checked	Returns true if the button has its *TwoState* property set to true. Otherwise returns false.
className	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
enabled	Logical value (default true) that specifies whether or not the button responds when clicked. When set to false, the operating system attempts to visually change the button with hatching or a low-contrast version of the bitmap to indicate that the button is not available.
separator	Logical value that lets you set a vertical line on the toolbar to visually group buttons. If you specify a separator button, only its Visible property has any meaning.
speedTip	Specifies the text that appears when the mouse rests over a button for more than one second.
twoState	Logical value that determines whether the button displays differently when it has been depressed and consequently sets the *Checked* property to true. Default is true.
visible	Logical value that lets you hide (*false*) or show (*true*) the button. Default is *true*.
Event	**Description**
onClick	"Action code" that executes when the button is clicked.

Chapter 8 Using Source editor

Chapter

8

Using the Source editor and other code tools

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

- The Source editor

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

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

- The Code Builder

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

- The Command window

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

Using the Source editor

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

Note

If you prefer to open instances of the Source editor in separate windows, rather than using one window with tabbed pages, you can set this preference in the Display page of the Source Editor Properties dialog box (With the Source Editor open, Properties | Source Editor Properties | Display).

To open the editor, do one of the following:

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

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

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

Starting with dBASE Plus 9 new features were added to make the Source Editor a much more powerful and useful tool. Some of these features include ...

- End of Line: (now know where the end-of-line mark is)
- Code Folding: (lots of code, reduce the length and focus on only the code you need)
- Indentation Guides: (help better document the code)
- Line Numbers: (Know the exact line number in the code and use in combination with bookmarks)
- Comment line: (Ctrl+Q to toggle commenting a line or uncommenting a line)
- Comment Block: (Ctrl+Q on selected lines)
- □□□□□□□□□ Make selection upper case: (Simple editor tool, make the selected text ALL UPPER CASE)
- □□□□□□□□□□Make selection lower case (Simple editor tool, make the selected text all lower case)

These new features can be changed or added using the Source Editor Properties dialog, by using the Edit and View menus or by using the Key Combinations.

Three-pane window with tree view

The Source editor is a Three-pane window:

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

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

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

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

The Search view pane (bottom pane) can be sized to make room for more or less of the code. Use the mouse to hover over the divider and move the line to view more or less of the top or bottom pane. The Search pane will show the results of searches when doing a Find in File search (Edit | Search | Find in File).

The Find in File search option uses a search engine to search through many files. The results in the Search view pane can then be used to open any file that is found with the search parameters. to open one of the files in the results just double click on the line in the Search pane and it will open the file in the Code view pane above.

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

Notes on the Source editor

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

- Compared to the former Method editor:

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

  The tree view points at the top and bottom of the source files, showing the equivalent of "header" and "general" in the former Method editor.
- When editing a .PRG file, the Method menu's New Method, Delete Method, and Verify Method commands are available. They work on whatever "method" is at the current cursor position. If no method can be identified, the menu commands are unavailable.
- When designing a form, report, menu, or data module file, three more commands are available on the Method menu: Edit Event, Link Event, and Unlink. Edit Event can generate wrappers for functions or procedures that are not yet part of the source, useful with the new tree view.
- If you attempt to edit a method in a base class, and you elect not to override that method in the derived class, dBASE PLUS opens the source file for that base class in the designer. If it is already opened, it is given focus, and the cursor is positioned at the method.
- When you switch focus from the designer to the editor, it purges the editor's Undo buffers.
- Opening a file named in code:

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

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

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

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

Creating a new method

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

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

Note

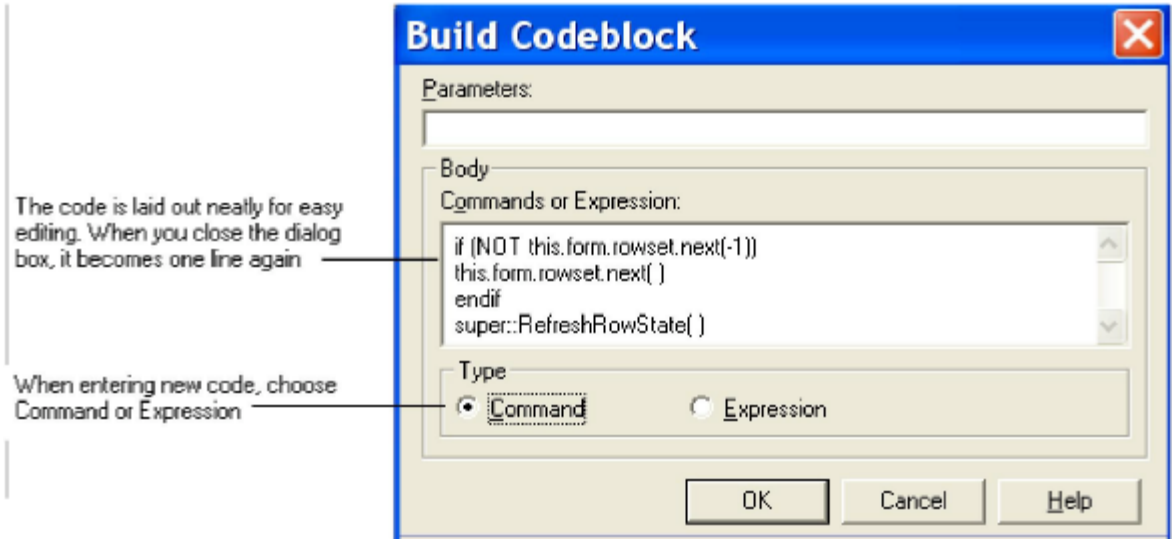
A method is a function defined in a class. The Form and Report designers are object-oriented; forms and reports are classes. Therefore all methods are defined and appear in the Source editor with the reserved word function, and are sometimes (loosely) referred to as functions.

The Code Block Builder for editing code blocks

A code block is a data type that can be stored in a variable or property. Code blocks are used in forms and reports to define events or text properties.

Because code blocks cannot span multiple lines, using the Source editor to edit a long code block can be cumbersome. So, when you choose to, you can open the Code Block Builder, which temporarily lays out the code one command per line, with the parameters in a separate text box.

Figure 0.1 Code Block Builder



Edit what you need to, and choose OK. The code block appears in your code as one line again.

You don't have to open the Code Block Builder if you don't want to. You can edit directly in the Inspector or the Source editor.

To create or edit a codeblock

To create a new codeblock for an event,

1. Select CodeBlock from the event's drop-down Type list.
2. Click the wrench tool beside the event to open the Code Block Builder.

To create a new codeblock for a Text control,

1. Select its *text* property in the Inspector.
2. Select CodeBlock from the *text* property's drop-down Type list.
3. Click the wrench tool beside the property to open the Code Block Builder.

Editing an existing code block

If you have an existing code block you want to edit (it will be in an event or the *text* property of a Text control), you can open the Code Block Builder dialog box in these ways:

- For an event,

  - Select the event in the Inspector, and then select the wrench tool beside it, or
  - Choose Method | Edit Event, and if a code block is already associated with the event, the Code Block Builder opens (otherwise, the Source editor opens).
- For a Text control, select its *text* property in the Inspector, and then select the wrench tool beside it.

Make your changes in the Parameters text box and in the Commands Or Expression text box.

When you click OK, dBASE PLUS checks the syntax of the code block. If an error exists, dBASE PLUS attempts to repair the error. If it can't, a warning message box notifies you of the error, and the Code Block Builder stays open so you can fix the error. Focus is placed on the text box where the error occurs: Parameters or Commands Or Expression. If you don't know how to fix the error, you can choose Cancel and dBASE PLUS keeps the previous code.

If the code block is error-free, then the Code Block Builder closes. The code block is condensed back into one line and displayed in the appropriate line in the Inspector. The indentations and carriage returns are removed.

The Command window

The Command window is used to directly execute one-line dBASE PLUS commands. It is handy for testing simple expressions and immediately seeing the results in the results pane. (It is the dBASE PLUS counterpart to the dot prompt found in dBASE IV and earlier DOS versions of dBASE.)

Note

The Command window is for temporary work only; you cannot save your work. However, you can copy the contents of the window or drag and drop to a source file. You can also print the contents (select what you want to print, and choose File | Print).

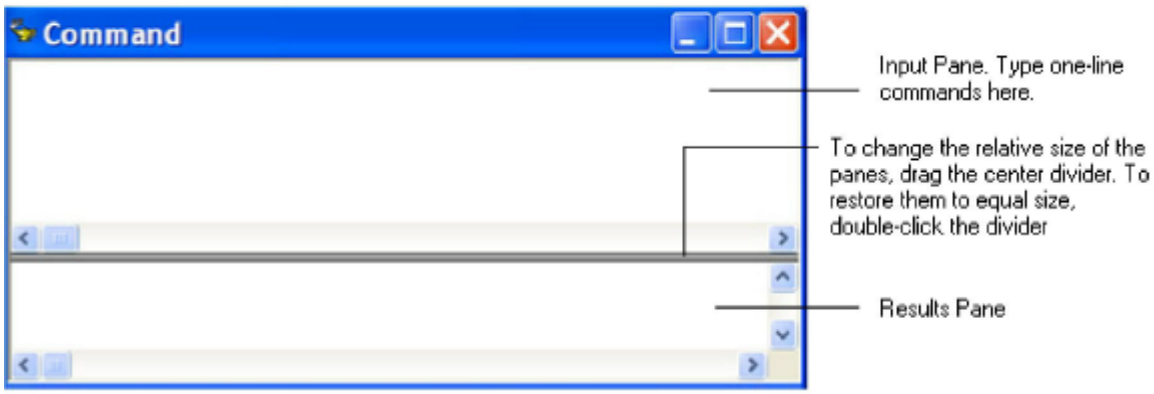
To use your own functions in the Command window, you must first load them:

```
set procedure to <filename> additive
```

To open the Command window, choose View | Command Window.

The Command window has two panes, as shown in the figure below.

Figure 0.1 The Command window



Command window panes have specific functions:

- The input pane is where you enter interactive commands You might use the input pane in this way if you find typing commands easier or faster than using the mouse and menus.

  The input pane echoes your actions in the dBASE PLUS interface, keeping a history of the commands you've executed. For example, when you create a new table by double-clicking the Untitled table icon, the Command window shows that you've executed a CREATE command.

- The results pane is where your command output appears, unless your commands create or call separate windows. It is also the default destination for the output of many programs. The results pane retains the last 100 lines.

To change the relative size of the two panes, drag the center divider. To restore them to equal sizes, double-click the divider.

To clear the contents of the input pane, close the window and select View | Command Window. To clear the results pane, choose Edit | Clear All Results.

Typing and executing commands


To execute a command, type it in the input pane and press Enter. You can also click the Execute Selection button on the toolbar or choose Edit | Execute Selection. You can delete commands like any other text. The commands you enter in the Command window remain there until you close the window or exit dBASE PLUS.

Because pressing Enter executes the command line, you must press the down-arrow key to enter more than one line into the Command window. The maximum number of characters per line is configurable in the Editor Properties dialog box. The maximum number of lines the input pane can hold is limited by virtual memory.

The command line defaults to insert mode, as indicated in the status bar. To switch between insert and overwrite modes, press the Ins key.

Executing a block of commands

In addition to typing multiple command lines, you can paste lines of command text from another source. You can also execute a block of command lines, provided the block does not contain nested structures or methods.

To execute more than one line of text in the input pane, select the lines with the mouse or use Shift and the arrow keys. Press Enter, or click the Run button on the toolbar, or choose Edit | Execute Selection.

Reusing commands

To reuse commands you've already entered in the input pane,

1. Scroll the window, if necessary, to display the commands you want.
2. Click the command line you want, or select a block of commands.
3. Execute the command (or commands) by pressing Enter, clicking the Run button, or choosing Edit | Execute Selection.

Editing in the Command window

Edit text in the input pane as you would in a text editor, using standard editing keys such as Backspace and Delete, and the Edit menu commands. Use the Edit | Search commands to search for and replace text in the input pane.

The command line is the line in the input pane containing the insertion point.

You can cut or paste code from Help or use commands from a program file by opening the file, copying the commands, and pasting them into the Command window. After the commands are in the Command window, you can test or modify them. The sample files provided with dBASE PLUS are a good source of working commands.

Saving commands into programs

If the input pane contains dBL code you want to use again, you can copy and paste it into a new program (.PRG) file or insert it into an existing program file.

You can also mark a block and choose Edit | Copy To File. dBASE PLUS displays the Copy To File dialog box so you can name the new file for the selected text. By default, the file has a .PRG extension, but you can change it to another extension. If you don't mark a block before you choose Edit | Copy To File, the entire contents of the Command window is selected.

Chapter 9 Debug

Chapter

9

Debugging applications

Debugging is the process of locating and eliminating errors—bugs—from an application. Use the dBASE PLUS Debugger to repair broken code and resolve problems in your forms, reports and programs.

With the Debugger you can

- Load and debug multiple files.
- Control program execution by stepping through an entire program line by line or skipping to defined breakpoints.
- Monitor the values of variables, fields, and objects. You can even make temporary changes for testing purposes, and then update your code using the dBASE PLUS Source editor.
- View subroutines (methods, procedures, and functions) that the main program calls, and track the points at which each is called.
- Stop program execution at any point, or run full-speed to the cursor position.
- Run the Debugger as a standalone application to debug compiled programs.

Types of bugs

The two most common types of bugs are syntactical and runtime errors.

Errors in syntax include such oversights as misplaced braces or *endif* statements, and are generally caught by the compiler before you even get to the debug stage. If you run uncompiled code through the Debugger, however, it will easily catch any syntactical errors.

Runtime errors, such as calls to non-existent tables, are also quickly exposed by the Debugger, which automatically halts at the offending reference.

When you are stopped by any error, you can either cancel or suspend further execution of the program, ignore the error and continue running the code through the Debugger, or note the problem, open your dBASE PLUS Source editor, fix the code, and then return to the Debugger to check for additional errors.

The third, and least obvious type of bug is an error in program logic, and these are not detected so easily. If, for example, your program includes a method that is supposed to execute after a certain event, but the event is bypassed, you may need to use all the debugging power described in this chapter to track down and correct the problem.

Using the Debugger to monitor execution

There are three ways you can use the Debugger to monitor how your program executes:

- Run the program locally from the dBASE PLUS integrated development environment (IDE). Running from the IDE is convenient for checking code syntax or various parts of your program while you develop it.
- Compile your application, then debug it by typing debug <programname.exe> into the Command window. This method provides a "real world" test, showing how your program accesses tables, for example. After running your tests, you can use the dBASE PLUS Source editor to make any needed adjustments.
- Run the Debugger as a standalone application, set breakpoints, then run your program from dBASE PLUS. When the program reaches a breakpoint, control is handed over to the Debugger.

General debugging procedure

This section gives you a quick overview of debugging procedures. The process is examined in greater depth in subsequent sections.

The Debugger is always available whenever you run into an error when in Run mode. To open the Debugger and deal with the error on the spot, you only need to click the Debug button in the error dialog. When the Debugger opens, you can then proceed from step 2 in the instructions below.

Alternatively, you can run it before running your program, then choose a program to debug. Here's how:

1. Start the dBASE PLUS Debugger by right clicking on a source file in the Navigator and choosing Debug. The program's code appears in the Debugger's Source window, under a tab with the file's name. If you open multiple programs, each appears under its own labeled tab.
2. You can then configure a number of options:
   - If you intend to pause the execution of the program at certain points or isolate a section of the code for test-fix-test cycles, set breakpoints by double-clicking the Stop Hand pointer at the left of the line before which you want a breakpoint.
   - Open any tool windows you intend to use, for example, to watch variables.
3. If you set breakpoints, or if any kind of error occurs
   - The Error Message box displays the dBASE PLUS error message. Click OK.
   - The Debugger's Source window appears. The offending line immediately precedes the blue-highlighted line.
   - Check for the more obvious and typical errors: a misspelling or missing punctuation or spaces.
   - Inspect your public or private variables and expressions by holding the cursor over a variable until a popup dialog box appears with the variable's current value. This can give you a clue about what went wrong. You can also view all variables in the Variable tool window or set watchpoints for particular expressions and monitor these in the Watch tool window. The debugger, however, won't find the value of a LOCAL variable.
   - If the form does not appear quite the way you intended when you created it in the Designer, try adjusting some of the display parameters in one of the tool windows. If this works, make the same changes permanently in the code by editing the program in the dBASE PLUS Source editor.
   - For other types of errors, return to the main dBASE PLUS Source editor, locate the offending line and make your corrections. Save the file, then restart your program to check the results.
   - Restart the program to check the results of your correction.
4. If your application initially appears stable and displays properly, proceed by testing each of its features, entering or editing values, submitting changes to the server, or requesting updates. Click each button and

interact with the program in every possible way. Errors generated from these interactions might require you to step into subroutines and again check variable values at each step.

Note

Selecting File | Exit from the Debugger menu, the Debugger closes and program execution continues. In prior dBASE versions, selecting File | Exit while in the Debugger, halted program execution and returned to the Command Window. In dBASE PLUS program execution is halted by using the Stop button on the Tool Bar.

Whenever you need to run the program again from the top, simply switch focus to the Debugger, then switch back to your main program window.

Debugging runtime applications

To debug compiled programs, simply run the Debugger as a standalone application, set breakpoints, then run your program from dBASE PLUS (not from the Debugger).

When the program reaches a breakpoint, control is handed over to the Debugger.

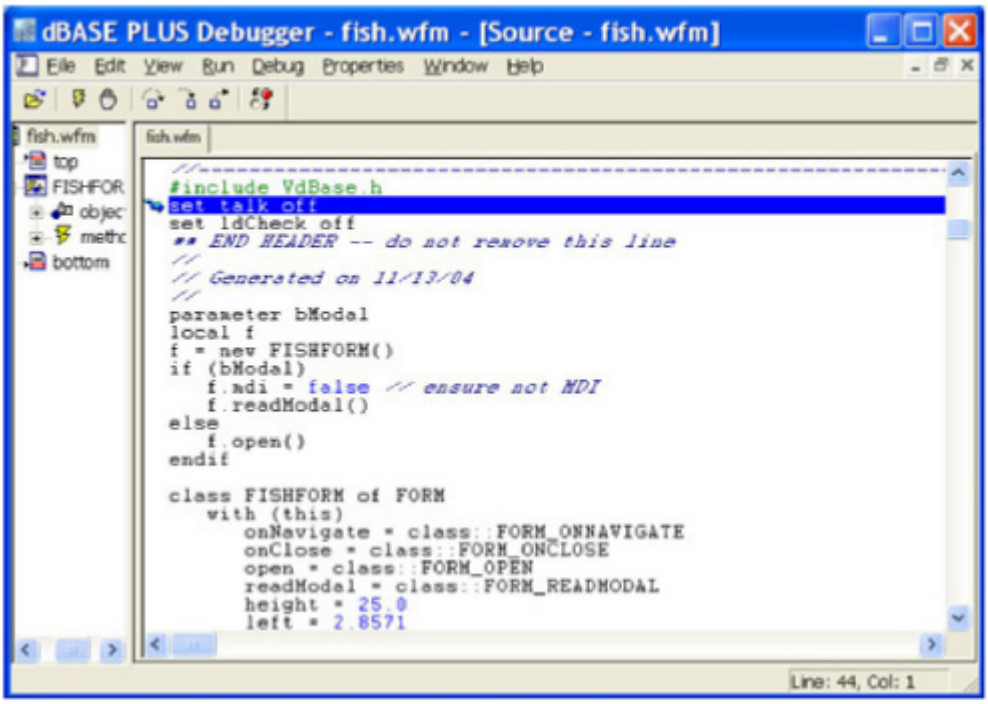
The Source window

The Source window is the main Debugger window. The code is read-only. The left pane of the Source window shows a hierarchical view of the objects in your code.

You can use the Source window to step through code execution line by line, to highlight breakpoints and identify the location of errors.

However, you cannot edit or repair your code in the Debugger's Source window; to do that, you have to use the dBASE PLUS Source editor.

Figure 0.1 Source Window



As your program runs, source code scrolls to the line about to receive program control. You can scroll the source code without affecting program execution; the next command to be executed remains highlighted regardless of where the cursor is in the Source window.

Your program runs in the background while the Debugger retains focus. You can interact with your program by moving focus to it (use Alt+Tab to switch to it, or minimize the Debugger and click your program window). While you interact with your program, its code continues to scroll in the Source window. If you close the application, your program's code remains open in the Source window.

You also use the Source window to locate and set breakpoints. If an error occurs, the Source window automatically scrolls to the offending line, highlighting (in blue) the line immediately following the offending line.

To correct any detected errors, return to the dBASE PLUS Source editor, load your file, locate the offending line (using the dBASE PLUS Source editor's line-numbering feature) and fix the error.

To locate and move to a line number in the Source window

1. Choose Edit | Go To Line from the Debugger menu or right-click the Source window and choose Go To Line from the popup menu.
2. Specify the number in the Go To Line dialog box.

Using Go To Line to move to a line number in the Source window doesn't affect program execution, which remains paused at the last line you stepped or traced to, or at the last executed breakpoint.

To find a text string in the current program file

1. Choose Edit | Find from the Debugger menu or right-click the Source window and choose Find from the popup menu.
2. Specify the text to find in the Find dialog box.

The search begins from the current position of the cursor and is case-insensitive. If the text is found, the Source window scrolls to the line containing the text, and the cursor moves to the beginning of the text. To find additional occurrences of the same text, choose Edit | Find Next from the Debugger menu or Find Next from the Source window popup menu.

The Debugger tool windows

The Debugger's View menu offers access to four tool windows, described below, that help you track program elements during a debugging session.

Variables

Lists variables found in the currently selected program. You can limit the extent of the variable search by deselecting options in the Debugger Options dialog box (File | Options).

Watches

The Watch tool window lets you specify particular variables, fields, and expressions that you would like to watch as the program code executes. The window shows the changing values for the watched items as you test the program (by clicking buttons, sending queries, and so on).

Call Stack

This tool window displays a list of subroutines called by the current program. Each call to a separate program file (or module) is displayed as it occurs, showing its line number, function name, and path name.

Trace

Displays the output that appears in the Output panel of the Command window.

Docking the Debugger tool windows

To dock a tool window in the Debugger, click the frame of the window and drag to the side of the Source window where you want to fix the window's position. The small tool window enlarges to fit the side or bottom of the Source window.

To undock a tool window, click the frame of the tool window and drag to the center of the screen until the small outline appears, then release.

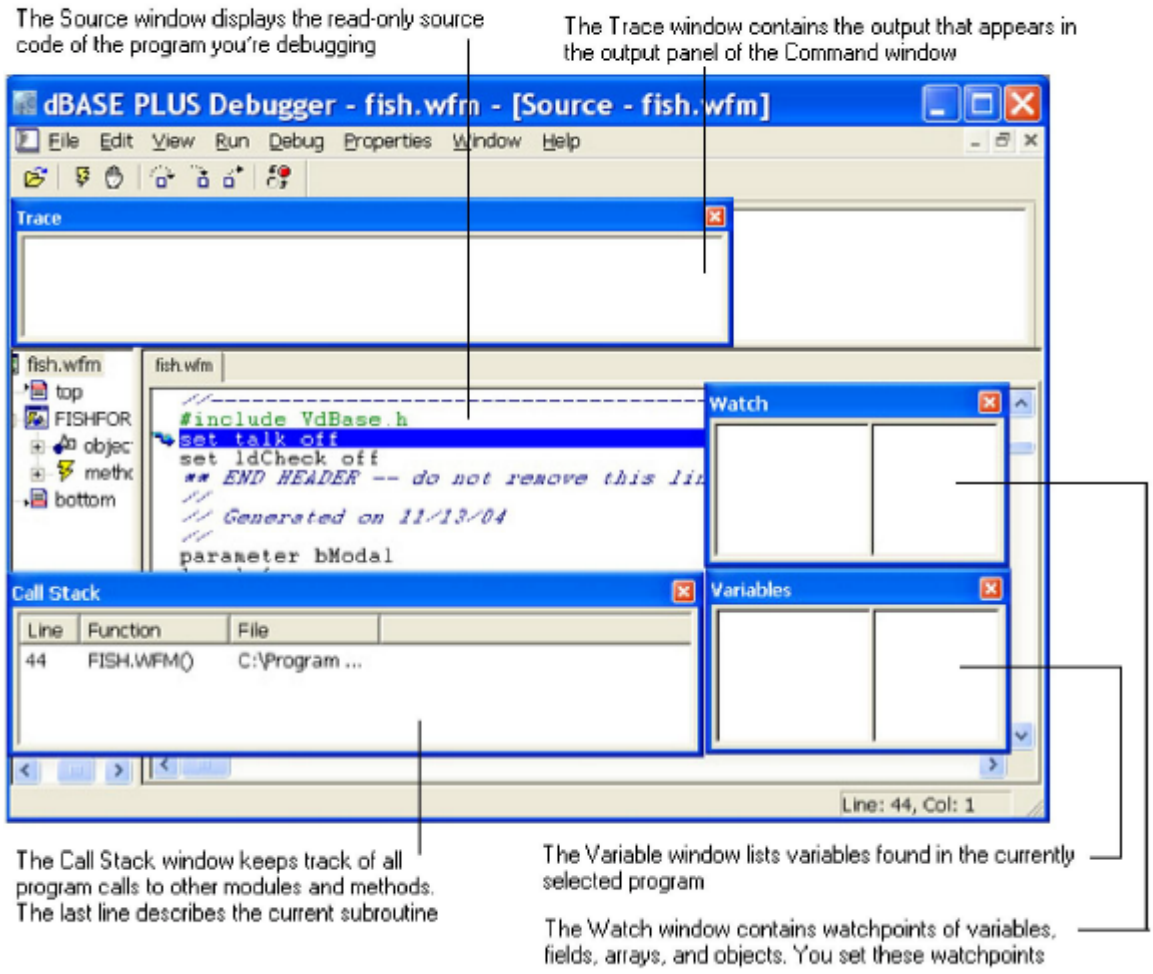
To disable docking, choose View | Tool Window Properties and uncheck the tool windows for which you want to disable docking. See Figure 9.2

Excluding variable types

To exclude variable types from the tracking process, choose File | Options to open the Debugger Options dialog box, then deselect items from the variables type group.

The Options dialog box also permits you to hide exceptions during a debugging session.

Figure 0.1 Debugger tool windows, docked



Controlling program execution

You can control program execution in the Debugger using the following commands and procedures, all of which are available from the Run and Debug menus.

Table 9.1 Methods of controlling execution in the Debugger

Method	Type of program execution control
Run	Runs the program, stopping at each error to display an error message. The line after the offending line is highlighted in blue.
Stop	Stops execution of the program.
Run To Cursor	Executes lines from the current position to the cursor location and stops there.
Step Over	Skips subroutines (any called functions, methods, or procedures).
Step Into	Shows line-by-line execution of subroutines, stopping at the end of each subroutine. You can step into another nested subroutine.
Step Out	Returns display of line-by-line execution to the preceding level of the program.
Break on next executable line	Causes program execution to stop at the next executable line, regardless of where execution starts.
Using breakpoints	Breakpoints are specific points in the program that you set to stop execution, letting you assess the situation. You can isolate a section of code for closer study by placing a breakpoint at the beginning and end of the section, then running that section repeatedly. You can further subdivide the section by adding more breakpoints.
Watching variables	You can see the current real-time value of any variable by holding the cursor over it. You can select particular variables (from the Variable tool window which shows all the variables in the program) and watch how their values change in the Watch tool window.

Stepping in the Debugger

Stepping executes your program line by line, pausing after each line so you can evaluate the result.

If you are confident about a block of code, you don't have to step through it again and again to get to the uncertain areas. You can, for example, step over any lines that call subroutines (including methods and expressions).

For example, if you have already determined that no bugs exist in a particular subroutine, select the line that calls the subroutine, click the Step Over button in the toolbar, and run the program. The Debugger steps over the execution of the subroutine so you can focus on the rest of the program. The call to the subroutine still occurs, and the stepped-over subroutine still executes; you just don't see the line-by-line execution in the Source window. The Debugger then stops at the line following the stepped-over subroutine.

On the other hand, if you want to check out what a subroutine is doing, you can *step into* it, and further, step into any nested subroutines as well. Then you can *step out* from the subroutine and return to the main program level.

Commands for stepping over, stepping in, and stepping out of subroutines are available from the Run menu and toolbar.

Using breakpoints

A breakpoint stops program execution so you can evaluate variables, fields, arrays, objects, and expressions; change the value of variables, arrays, and objects; and check what subroutine the program is in. Using breakpoints lets you run the program at full speed until it comes to a problem area; breakpoints give you an alternative to stepping through the entire program.

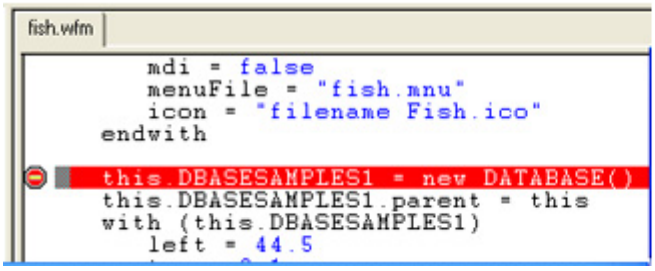
When you step or trace through a program, you're essentially breaking at each line. However, once you are certain that no bugs exist in certain parts of your program, there is no need to repeatedly step through each line; instead, set breakpoints at crucial places where the code is less certain, then run the program at full speed and evaluate program values at the breakpoints.

If, for example, you suspect a bug in occurs at one particular place, such as when a subroutine is called, you could set a breakpoint at the line that calls the suspect subroutine. You could then *step into* the called method or function.

Setting and removing breakpoints

To set a breakpoint, select a line of code in the Source window and either

- Move the pointer to the left of the command line where you want to enter a breakpoint. When the pointer changes to a Stop sign, double-click. The line is then highlighted in red.



  To remove the breakpoint, double-click the highlighted line again.

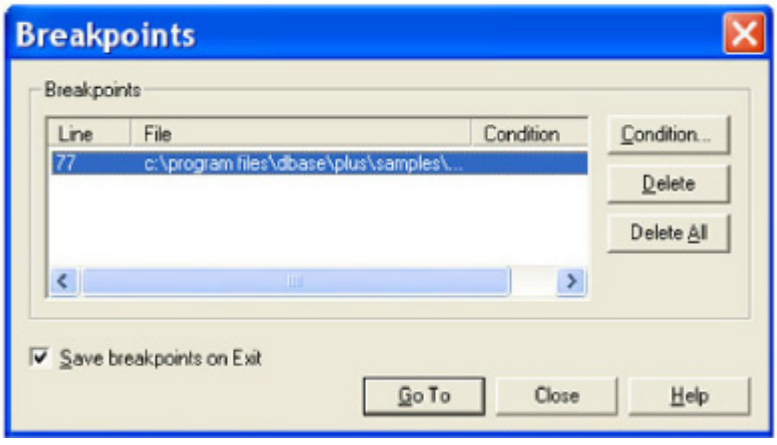
- Press Ctrl+B to add a breakpoint to the selected line of code or to remove a breakpoint from the selected line.
- Choose Debug | Toggle Breakpoint from the Debugger menu (or from the Source window's popup menu).

To remove all current breakpoints, choose Debug | Delete All Breakpoints.

Working with breakpoints

To see a list of breakpoints in a program, choose Debug | Breakpoints.

Figure 0.1 Breakpoint window



You can use the Breakpoints list to keep track of existing breakpoints in all open modules. The line number of each breakpoint is listed next to the path name of the program in which it appears.

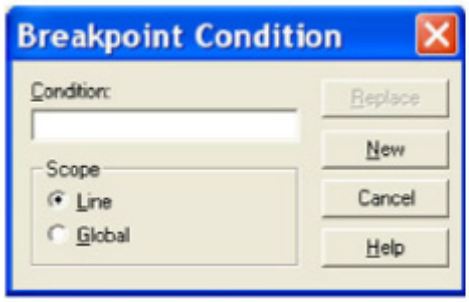
To delete a breakpoint, select it and click the Delete button.

To delete all breakpoints, click the Delete All button.

To go to any breakpoint, simply double-click the line number of the desired breakpoint (or select it and click Go To). The Breakpoint window closes and the Source window opens to the selected breakpoint.

To conditionally set breakpoints, click Condition to open the Breakpoint Condition dialog box, and type in an expression—such as the value of a variable, a global condition, or any conditional expression that defines a condition in which the current breakpoints should be active. If the condition is not met, breakpoints are ignored.

Figure 0.1 Breakpoint Condition dialog box



- **Line** Click the Line radio button to specify a condition for the line currently selected in the Breakpoint window. The conditional expression you enter applies only to the selected breakpoint line.
- **Global** Click the Global radio button to enter a conditional expression for the entire program. For example, you might have noticed that when a global variable reaches a certain value, say 10, the program becomes unstable, but until then everything works fine. To test your observation, you could set the condition *x=10* to force all breakpoints to activate when the global variable *x* reaches 10.

Running a program at full speed from the Debugger

The alternative to stepping and tracing, which examine your code line-by-line, is to debug at full speed. This lets you skip areas that you know are bug-free and concentrate on suspected problem areas.

If you set breakpoints in your program, you can debug at full speed and execute to the first breakpoint. You can then decide whether to continue running the program from the breakpoint or to proceed by stepping over or into subroutines.

To debug at full speed, either

- Click the Run toolbar button
- Choose Run | Run from the Debugger menu, or
- Press F9

Running to cursor position

You can also run at full speed until execution reaches the current cursor position in the Source window. To try this approach, choose Run | Run To Cursor.

Stopping program execution

In addition to using breakpoints and stepping techniques, you can halt execution of a running program any time by

- Clicking the Stop button on the toolbar
- Choosing Run | Stop from the Debugger menu

Debugging event handlers

You can use the same basic techniques to debug event handlers as you do for any other code sections.

These are the general steps for approaching error handlers:

1. Open the Debugger and load the program containing the event handler code.
2. Set a breakpoint.

3. Run the program, either at full speed, or by stepping or tracing through it.
4. When the program opens a form, the form gets focus. Interact with the form to trigger the event associated with the event handler. For example, if the event handler is an *onClick* method for a pushbutton, click the pushbutton.

When the event handler reaches the line or condition specified by the breakpoint, execution stops and focus returns to the Debugger. You can then inspect, step, or perform other debugging tasks.

Viewing and using the Call Stack

The Call Stack window tracks calls made to methods or functions, whether inside or outside of the running program. The list includes the line number, function name, and file name to which the calls were made.

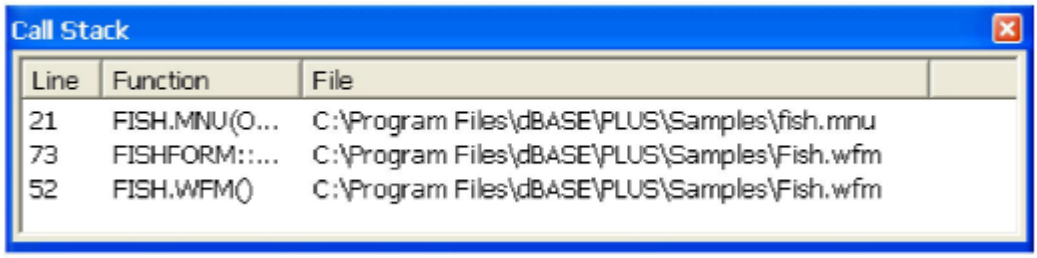
The last line in the Call Stack list is always a reference to the main program level.

Note that if you step through or over nested subroutines, calls within the nested subroutines are removed when execution steps out again.

Otherwise, the Call Stack list is updated whenever program execution pauses.

You can also use the Call Stack list to quickly shift Source window focus to any subroutine call. To do that, select a subroutine in the Call Stack, right-click, and choose Go To Source Line from the popup menu. Note that though the Source window is refocused to the calling line, the current execution point remains where it was, and will continue from that point if you resume program execution though the Source window is refocused to the calling line, the current execution point remains where it was, and will continue from that point if you resume program execution.

Figure 0.1 Call Stack window



Watching expressions

The Debugger's Watch window lets you monitor expression execution and results. You can even use it to temporarily change and retest variables. It can help you detect such problems as the assignment of incorrect values to variables, or assignment of values at the wrong points.

Watchpoints are evaluated and their current values displayed in the Watch window whenever program execution is paused.

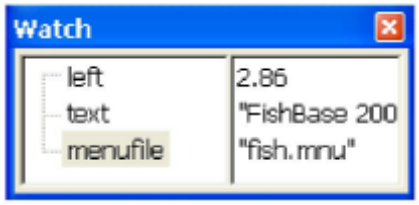
Adding watchpoints

To add a watchpoint, do one of the following:

- Choose Debug | Add Watch from the Debugger menu.
- Right-click the Watch window and choose either Add Watch or Watch Variable At Cursor from the popup menu.
- Press Ctrl+W.

The selected expression appears in the Watch window.

Figure 0.1 Watch window



Note

If you haven't already run the program past a point where variables, fields, arrays, or objects in the selected expression are initialized, "unknown" is displayed to the right of the expression.

Editing watchpoints

To edit an existing watchpoint,

1. Select the watchpoint in the Watch window.
2. Choose Debug | Edit Watch Value or Debug | Edit Watch Name from the Debugger menu. Alternatively, right-click the Watch window and select one of the editing commands in the popup menu.

Changing watchpoint values

In addition to watching expressions, you can temporarily change the value of a variable. If, for example, a variable is receiving a correct value, but at the wrong point, you could

- Add a watchpoint for the variable.
- Pause program execution by setting up a breakpoint at the line where the correct value is supposed to be assigned to the variable.
- Use the Watch window to directly assign the correct value to the variable.

It's important to remember that this sort of testing does not result in a permanent code change. It is only intended to let you examine how your program behaves when the correct variable value is assigned. If the test is successful, you can open the dBASE PLUS Source editor to permanently correct the part of the program that was responsible for returning the wrong value.

Chapter 10 SQL Query Builder

Chapter

10

SQL Query Builder

Starting with dBASE PLUS 8, the new SQL Builder feature is one that dBase believes will offer years of expanded capability and functionality. One of the main reasons for replacing the older SQL Designer is that we needed to be able to add additional databases quickly and easily. The new SQL Builder supports the latest databases, in a fast, consistent interface, with additional room for enhancement and upgrades.

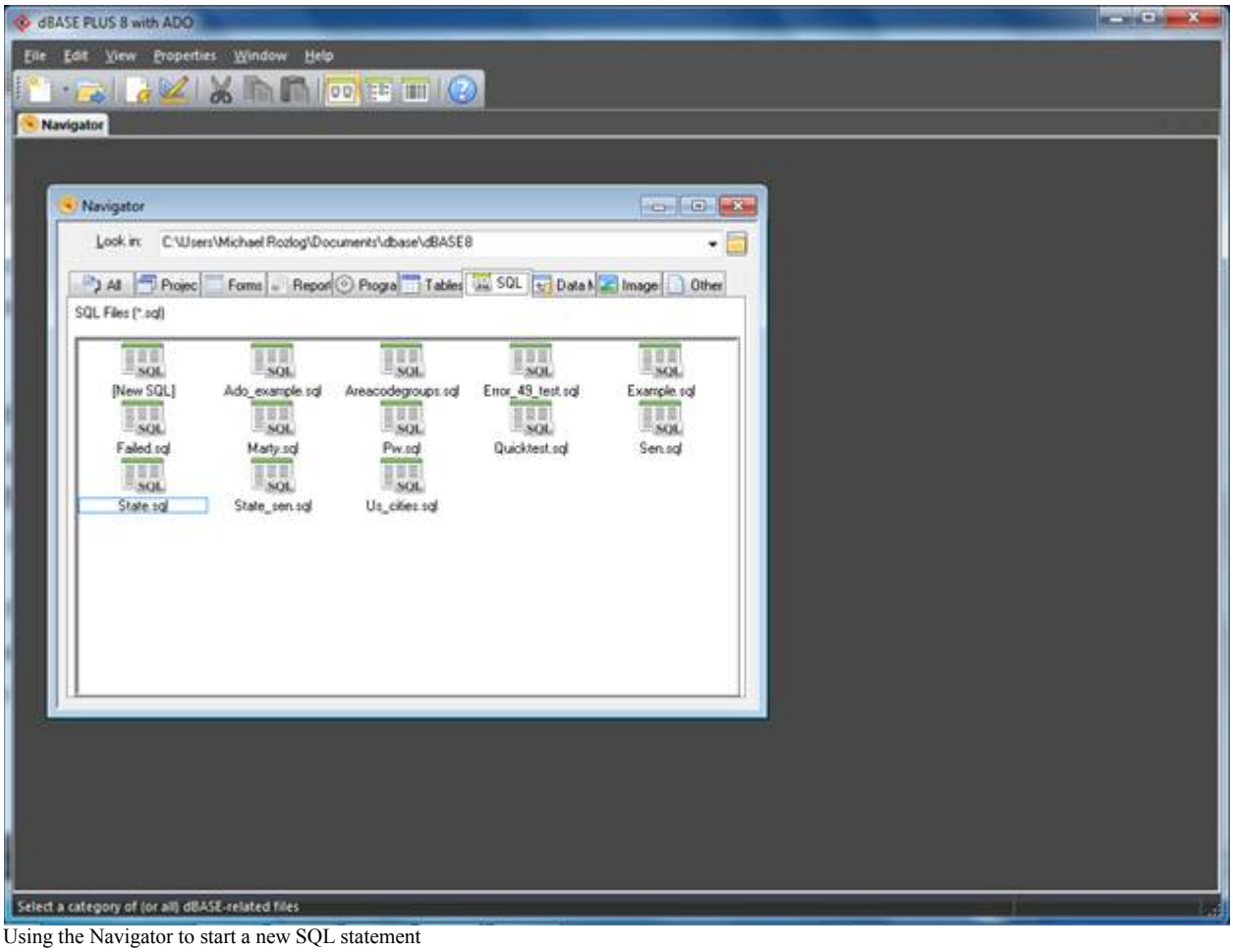
The new SQL Builder also gives much more control over the type and style of the SQL being generated and saved. The SQL Builder allows developers to create very simple, to very deep and difficult, SQL right from the drag-and-drop interface, which will be covered later in this documentation.

Keep in mind, to work with SQL Builder, you need basic knowledge of SQL concepts. SQL Builder will help you to write correct SQL code while hiding the technical details, but only by understanding the SQL principles will it be possible to achieve the desired results.

Selecting a database access method (ADO or BDE)

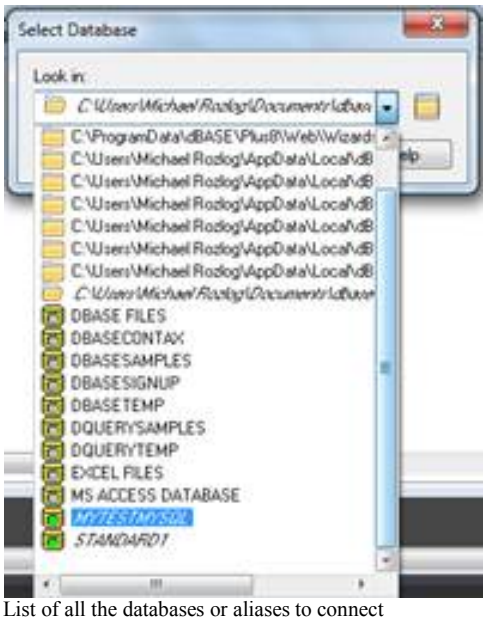
The first step in using the new SQL Builder is picking the data-access method to be used, either ADO or BDE. Note: At this time, SQL Builder does not support mixing different data-access methods from within an SQL statement.

The first thing to do is pick the SQL tab on the Navigator or click the File|New|SQL from the menu system. Below is an example of the Navigator being used to create a new SQL statement using the SQL Builder:



Using the Navigator to start a new SQL statement

Once the Navigator is on the SQL tab, you can select the [New SQL] in the top left corner of the Navigator. Double-click and you will be offered a database connection dialog will appear:



List of all the databases or aliases to connect

In the above example notice that there are two different listings; one states STANDARD1 and the other is MYTESTMYSQL. Each of these two items represent a different data-access layer. The STANDARD1 is using the BDE to make connections while the MYTESTMYSQL is connecting to a remote-cloud server running MySQL.

Therefore, if the developer selelcts the STANDARD1, then the BDE will be used for database access. If on the other hand, the developer chooses MYTESTMYSQL then ADO will be used. Note: While ADO is being used, the MYSQL database is actually using the ODBC driver socket that is part of the ADO technology.

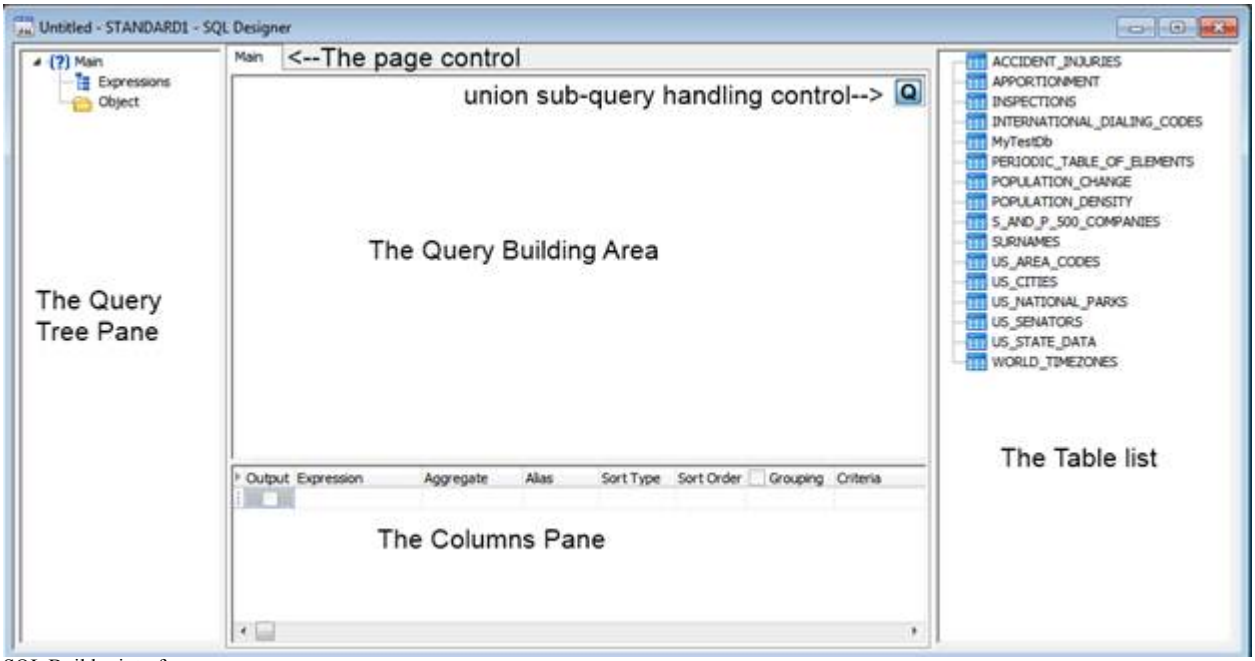
Connecting to the BDE

Connecting to ADO

Once you choose the data-access technology, press the OK button to continue and the rest of the SQL Builder interface will be loaded with the tables associated with the selection of the data-access technology.

2-Way SQL Development

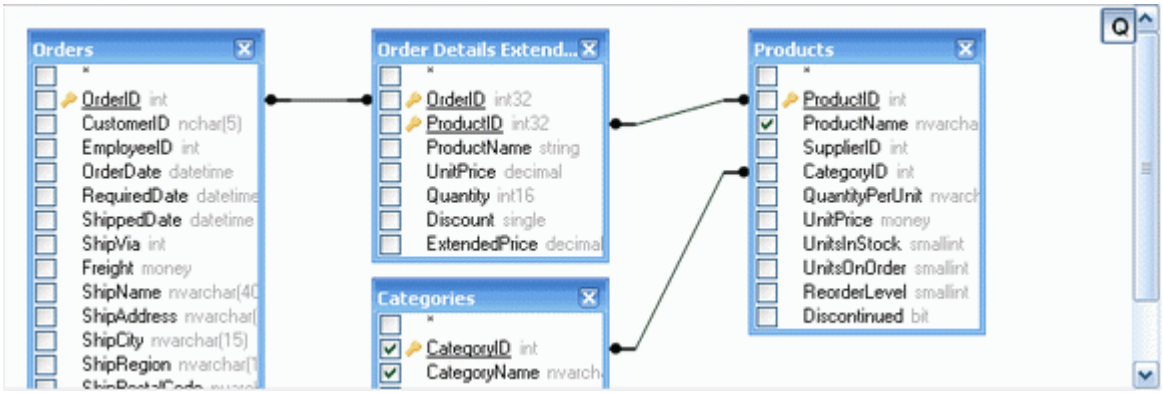
The first thing you should notice when starting up SQL Builder is that it fits the dBASE development paradigm rather well. It supports the concept of 2-Way development. If you don't remember what 2-Way development is, it is the process if you make changes on the GUI designer, the code will automatically reflect those changes, and if you make changes to the code, the GUI designer will automatically show those changes in real-time.



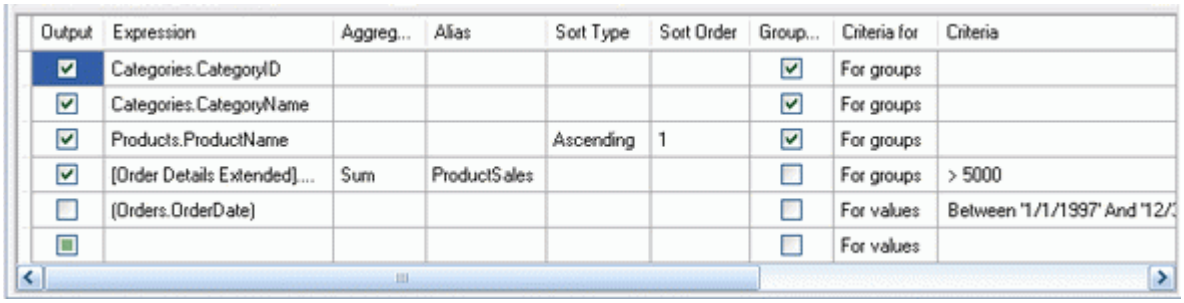
SQL Builder interface

The main window of SQL Builder can be divided into the following parts:

- **The Query Building Area** is the main area where the visual representation of the query will be displayed. This area allows you to define source database objects and derived tables, define links between them, and configure the properties of tables and links.

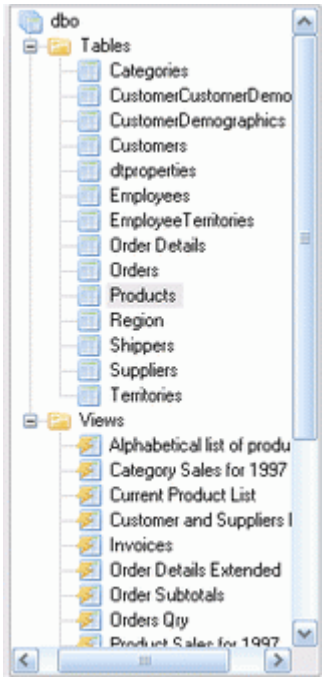


- **The Columns Pane** is located below the query building area. It is used to perform all the necessary operations with query output columns and expressions. Here, you can define the field aliases, sorting and grouping, and define criteria.

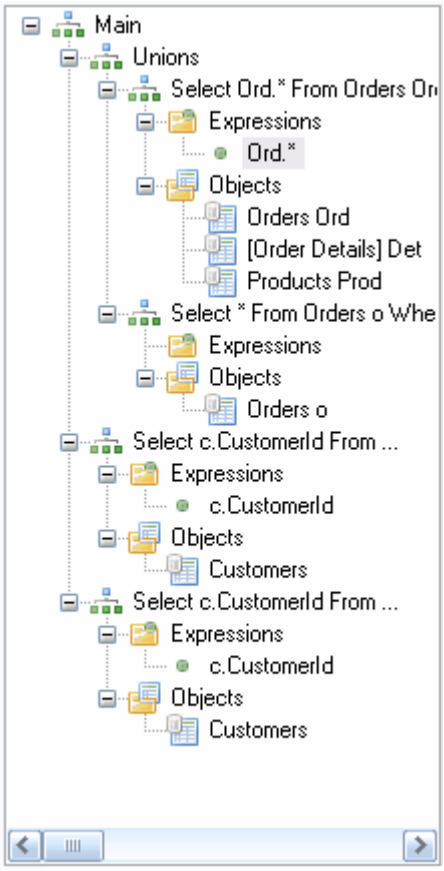


Output	Expression	Aggreg...	Alias	Sort Type	Sort Order	Group...	Criteria for	Criteria
☑	Categories.CategoryID					☑	For groups	
☑	Categories.CategoryName					☑	For groups	
☑	Products.ProductName			Ascending	1	☑	For groups	
☑	[Order Details Extended]....	Sum	ProductSales			☐	For groups	> 5000
☐	(Orders.OrderDate)					☐	For values	Between '1/1/1997' And '12/
■						☐	For values	

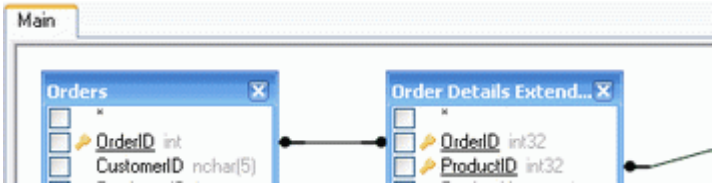
- **The Table list** is located at the right. Here, you can see and browse your table's objects.



- **The Query Tree Pane** is located at the left. Here, you can browse your query and quickly locate any part of it.



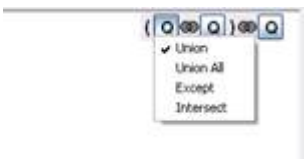
- **The page control** above the query building area will allow you to switch between the main query and sub-queries.



- The small area in the corner of the query building area with the **"Q"** letter is the **union sub-query handling control**. Here, you can add new union sub-queries and perform all the necessary operations with them using the popup menu.

Union Sub-Query operations

Operations with brackets



Select union joining operator

Again, at any time, the developer can see the SQL code behind the GUI based SQL Builder by simply pressing the F12 key to toggle between graphic view and code view. This can be done at any time during the development process. Again, remember that any changes to the GUI will change the code and any code changes will show in the GUI representation. This will be highlighted below:

This is a SQL base code

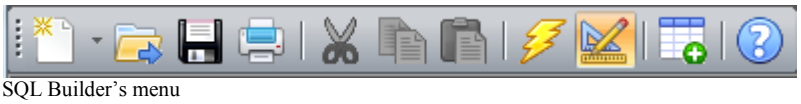
Clicked the Surnames, notice the code difference

Now change the code back to the "Select * from SURNAMES" and look at the difference in the GUI, as shown below:

Remove the SURNAMES.* from the code

Notice the check in the GUI is missing from the * in SURNAMES table

Finally, we should highlight the menu that is associated with the SQL Builder.



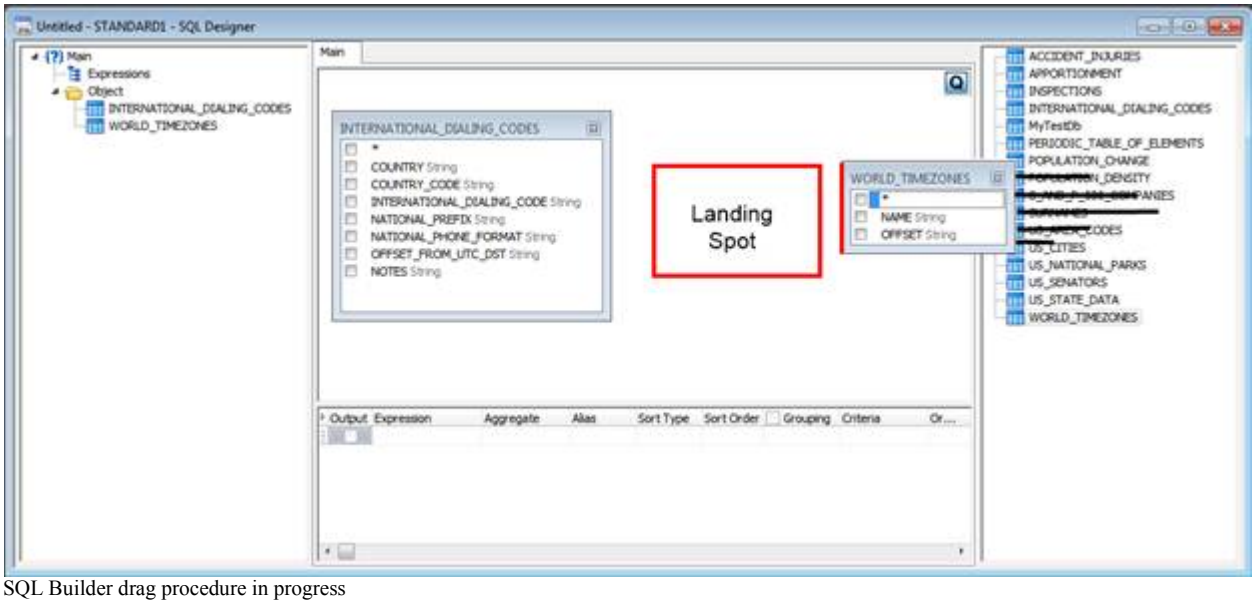
SQL Builder's menu

Starting from the left:

- **New –** this gives you a dropdown to pick or create a "new" something
- **Open –** this will open and load a file
- **Save –** this will save a file
- **Print –** this will print the active file
- **Cut –** this is the cut for the selected editor for the clipboard
- **Copy –** this is the copy for the selected editor for the clipboard
- **Paste –** this is the paste for the selected editor for the clipboard
- **Execute –** run and display the results of the SQL statement
- **Design –** this will return the product to the Design surface, the Execute and Design can toggle
- **Add Table –** this will add a table or allow for another database alias to be picked, this can also be done with the hotkey {CTRL-A}
- **Context Sensitive Help –** this will take you to the help associated with the particular in focus part. For example, if you are on the Select statement, then press Help will start on the Select statement outline

Drag and Drop execution

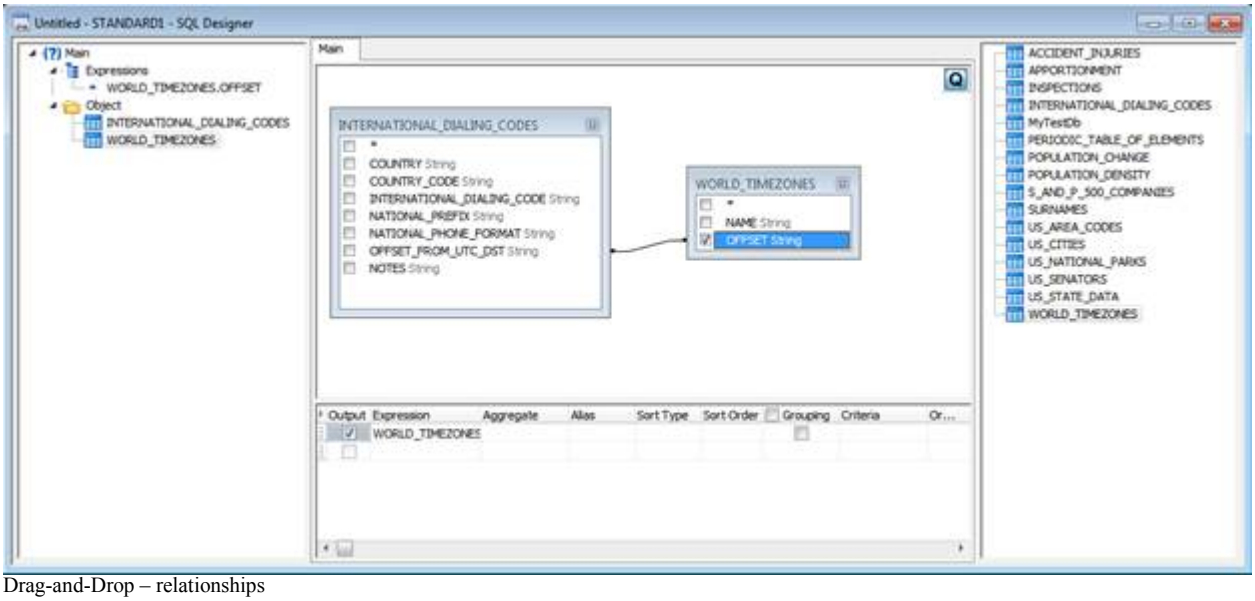
SQL Builder allows for full drag-and-drop functionality. Use the mouse to select a Table from the Table list on the right side of the interface and hold down the left-mouse and drag the table onto the Query Building area as shown below:



SQL Builder drag procedure in progress

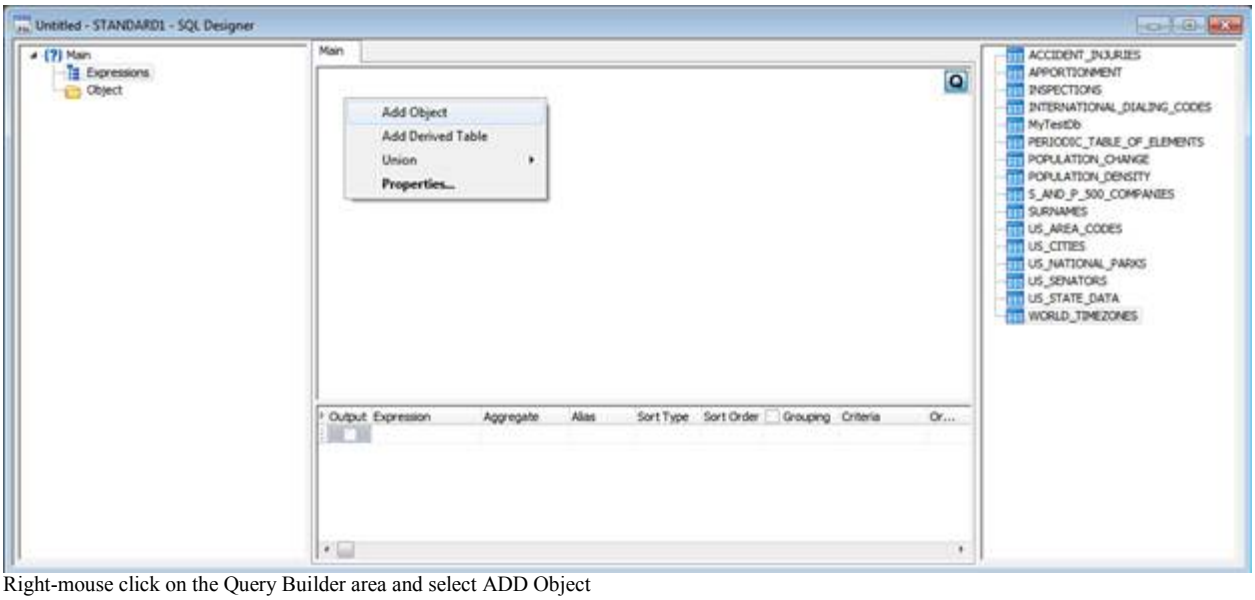
Once you have the table in the desired location, release the left-mouse key and the table will be placed in that exact spot.

The tables in the Query Builder area can be moved by using the same drag-and-drop technique used to put tables on the Query Builder area. While the query relationships will be covered later in the documentation, the drag-and-drop features area is also used to create relationships between tables as shown below:



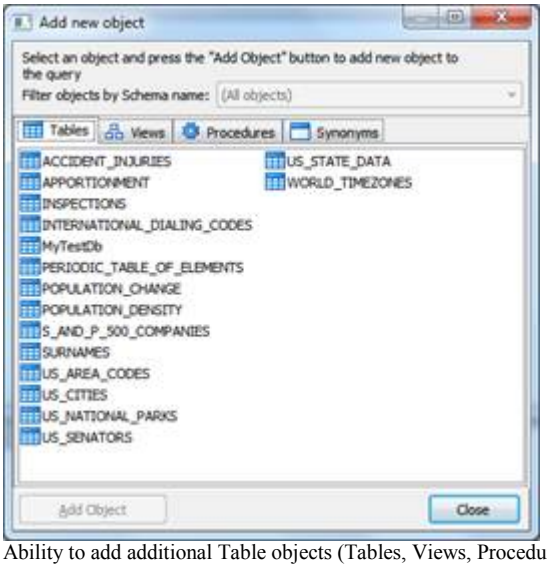
Drag-and-Drop – relationships

In the Query Building area, there is another way to add Tables objects to the designer surface. Right-mouse click on the Query Building area and the following dialog will be displayed:



Right-mouse click on the Query Builder area and select ADD Object

Once selected, the Add Object should give the following dialog:



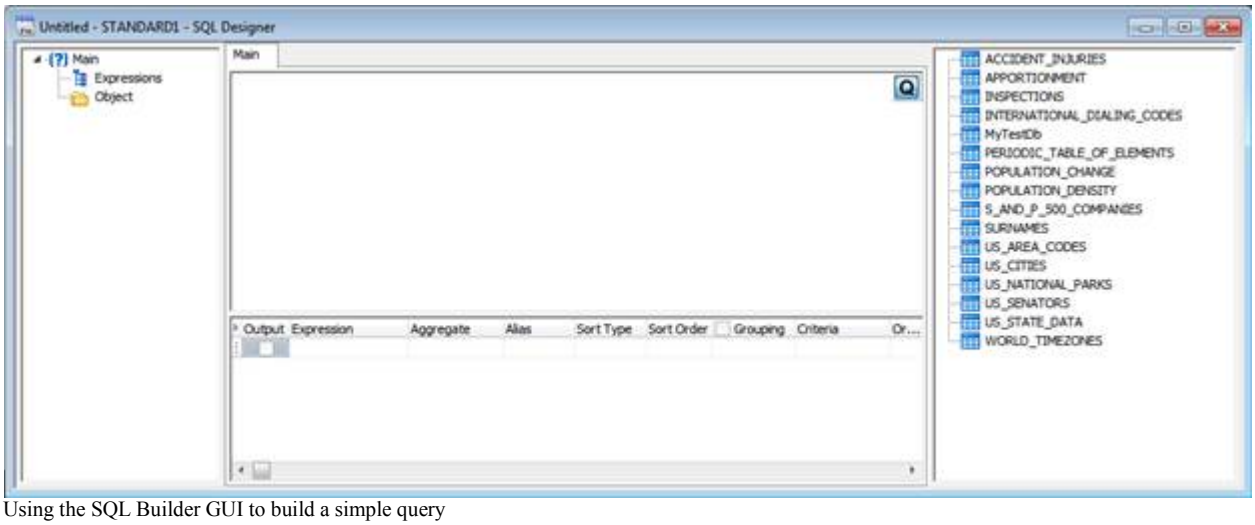
Ability to add additional Table objects (Tables, Views, Procedures, and Synonyms)

Select the desired object and click the Add Object button to add it to the Query Builder area. You can select one or several objects by holding the Ctrl key and then press the Add Object button to add these objects to the query. You can repeat this operation several times. After you finish adding objects, press the Close button to hide this window.

Simple Query

SQL Builder can be used to create any kind of SQL, from the very basic to the very complex. How complex? SQL Builder allows you to build complex SQL queries with Unions, Sub Queries, and Derived Tables visually. In this section, the steps needed to perform a simple query will be outlined.

To start, click on the Navigator and select the SQL Tab. There you will see a [New SQL] item and double-click on that time. This will display the Select Database dialog; pick a data connection that has tables or databases associated with it. After clicking the OK button this will start the SQL Builder graphical user interface, as shown below:



Using the SQL Builder GUI to build a simple query

For this example, the database picked will be the US_SENATORS. This is a small database and is very quick to show the results. Click and drag a table from the Table List on the right and drop it in the Query Builder area in the top-center. Notice the table is represented by a box with all of the defined columns displayed as below:

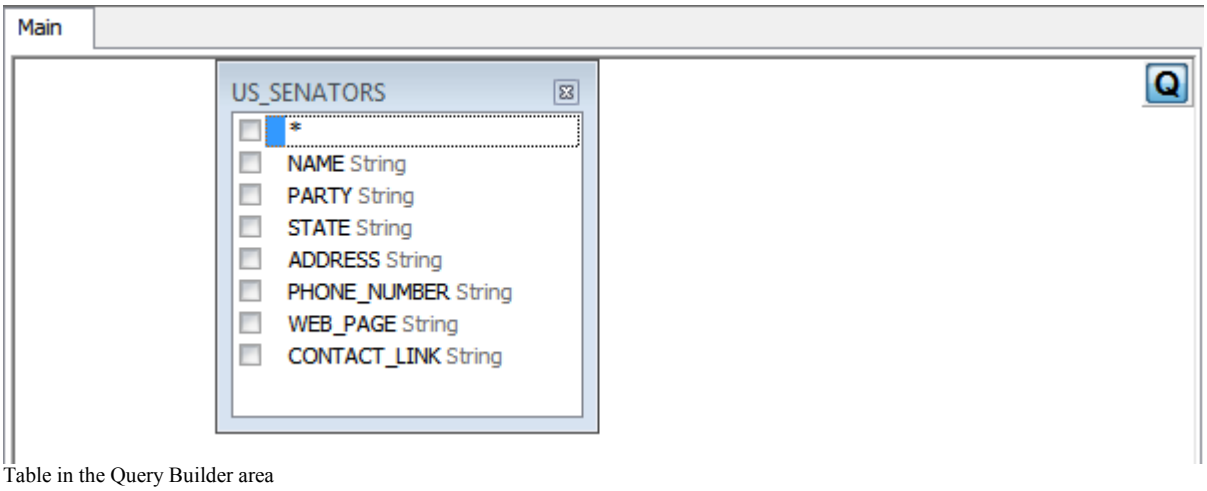
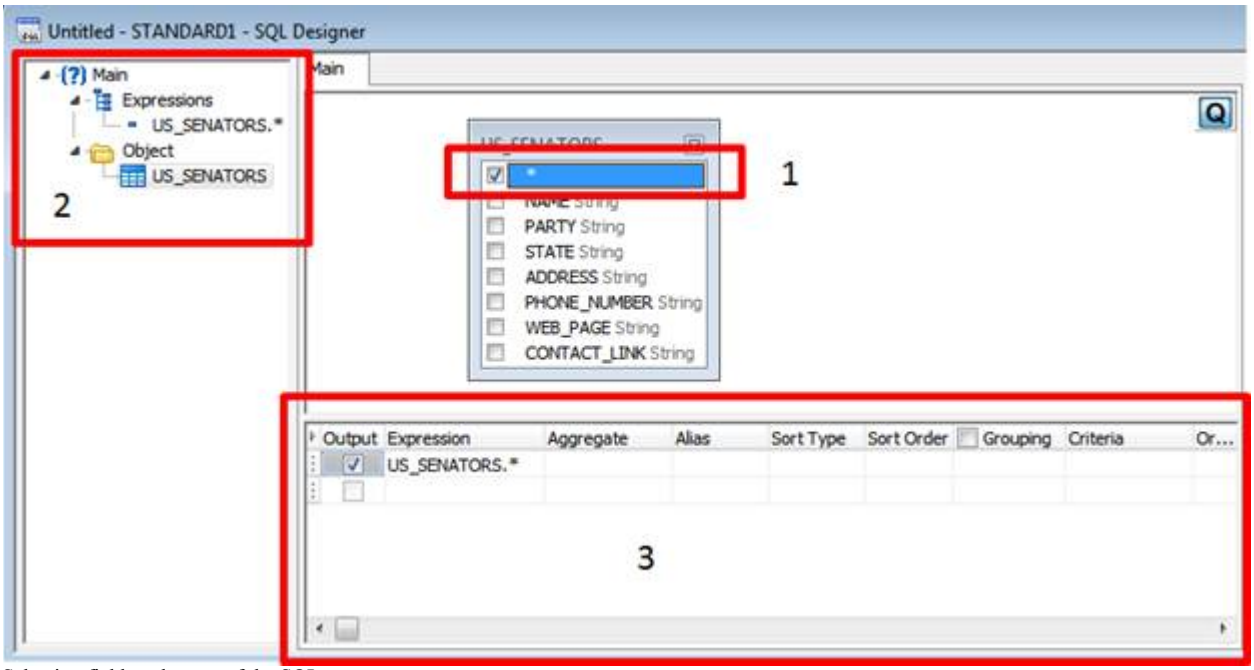


Table in the Query Builder area

The first thing that needs to occur in the example is choosing which fields you want to display. This is a Simple click procedure in SQL Builder. If you want all of the fields to show in a particular Table, you can just click the top-level "*" as shown below:



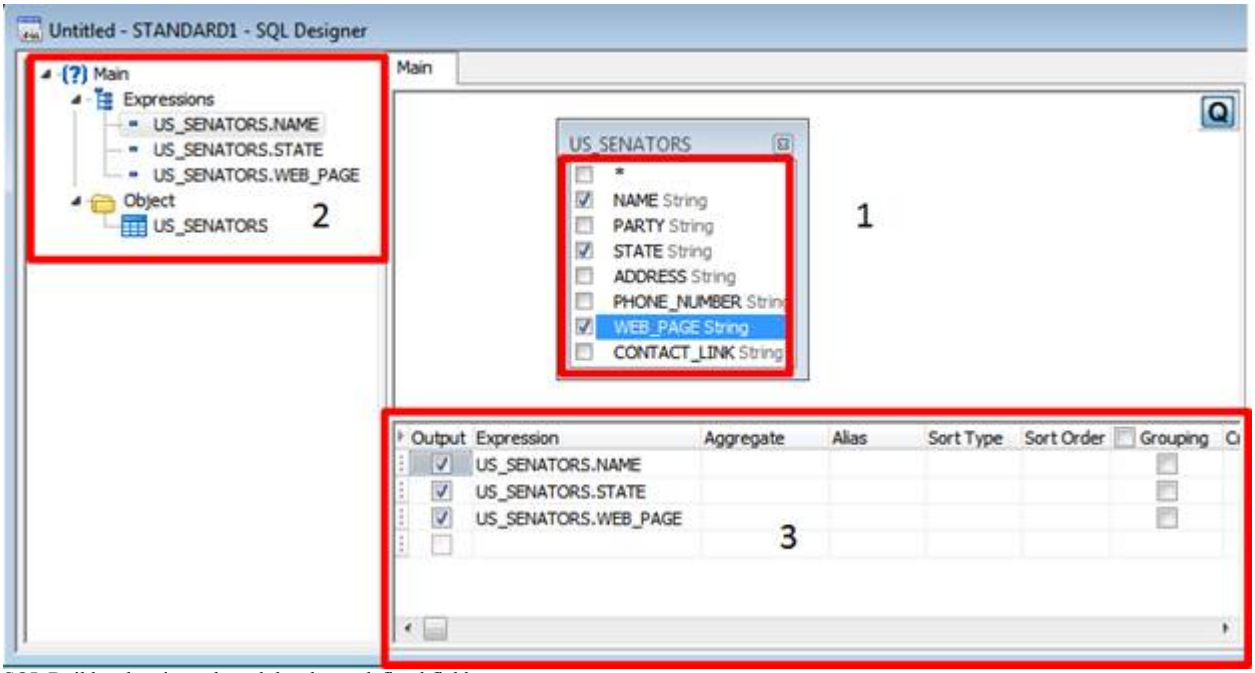
Selecting fields to be part of the SQL statement

There are 3 main areas that should be looked at in the above image. The first area is the table. Notice that it has the checkmark beside the "*" field, which means all fields will be included in the SQL statement.

The SQL code generated: Select US_SENATORS.* From US_SENATORS

The next area is the Query pane on the left that outlines the SQL being generated. Finally, review the contents of the Columns pane (number 3), where it also highlights which fields have been selected.

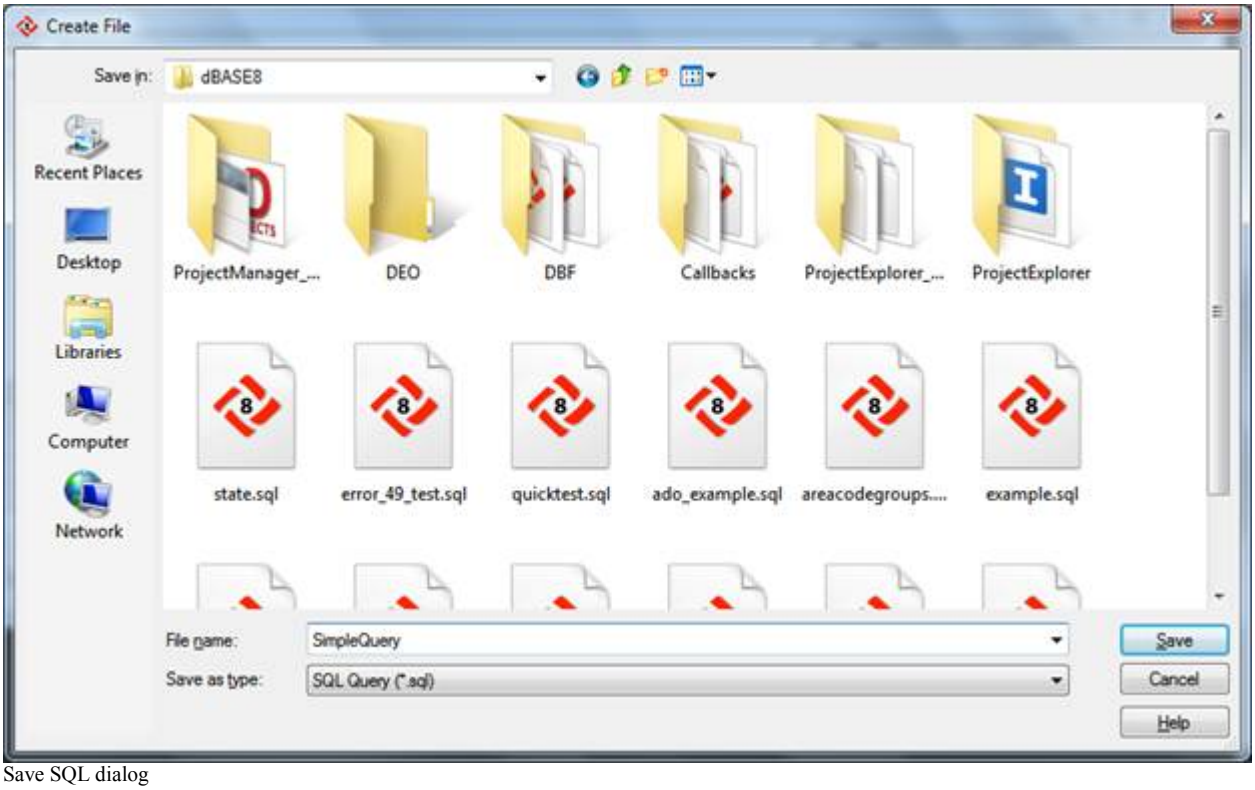
If you only need a few of the fields, instead of selecting the "*" field, you can click the desired field and a checkmark will be display beside that particular field, as shown below:



SQL Builder showing selected developer-defined fields

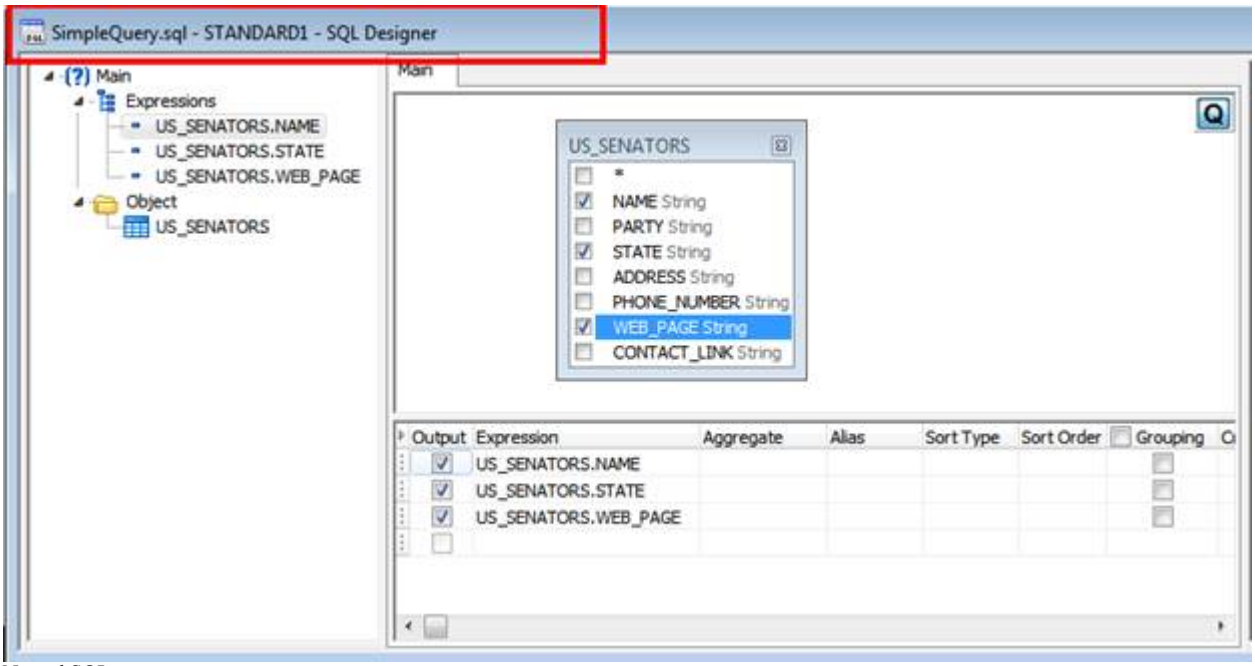
In the #1 pane, the selected fields have a checkmark beside them, the #2 pane shows each field in the SQL statement, and the #3 pane is showing the items ready for advanced SQL features. These field selections can be changed at any time during the development process or later when loaded back into the system.

Once the simple SQL has been defined, the next steps in going through the SQL development process is to Save the SQL statement. For a more in-depth review of the Save procedure, please refer to the "Saving an SQL statement" later in the SQL Builder documentation. The fastest way to save is to click the Save icon on the toolbar, which will display the Save dialog. Name the SQL and hit the OK button, as shown below:



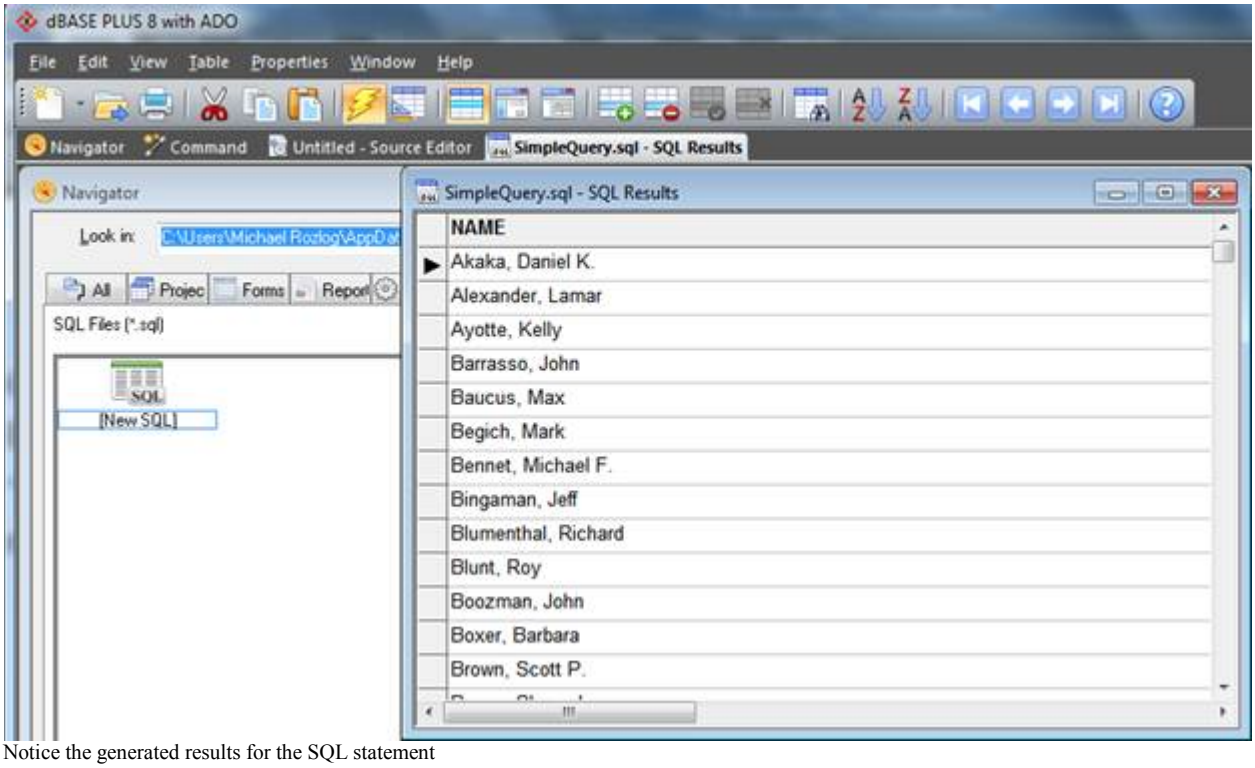
Save SQL dialog

The dialog will disappear and the SQL will now be named and will show in the interface:



Named SQL query

Once the SQL has been saved, it can now be executed by clicking the Execute toolbar button. For a more in-depth review of the Execute procedure, please refer to the "Executing an SQL statement" section later in the SQL Builder documentation. A new window with the results of the SQL statement will be generated:



Notice the generated results for the SQL statement

Please note that when executing an SQL statement, the SQL Builder will disappear. In addition, when the Results window is clicked, the associated toolbar will be changed as well. When in the Results view, you have the ability to change or modify the contents of the data being displayed.

If you are fine with the results, or notice additional changes need to be made, you can simply click the Design toolbar button and the SQL Builder GUI will reappear and the Results window will be closed automatically.

Those are the steps for creating a simple query using the SQL Builder found inside dBASE PLUS 9.

Executing an SQL statement

In dBASE PLUS 9, the Execute and Design toolbar buttons in the Execution toolbar are closely tied together:

Execute and Design toolbar buttons

Any time after the SQL has been saved, you will have the option to "Execute" the SQL statement. By clicking on the Lightning Bolt button. The "Execute" process can also be activated using the View|SQL Results menu item on the main product menu, or alternatively it can be activated with the F2 keyboard shortcut.

Once the results have been reviewed, you can return to the SQL Builder design surface by clicking on the "Design" toolbar button to the right of the "Execute" toolbar button. This will close the results window and return you back the SQL design surface. This operation can also be activated using the View|SQL Design menu item on the main menu.

You can go-back-and-forth between Execute and Design.

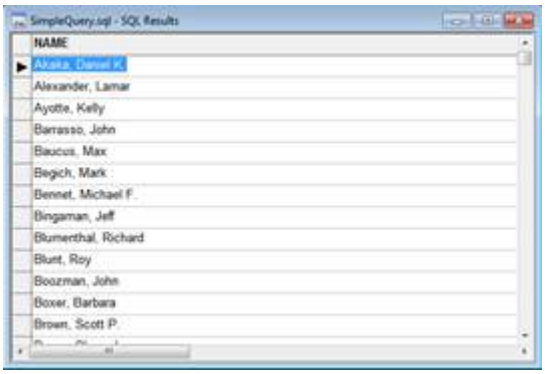
During the "Execute" phase, when the Results window has focus or is clicked on, the toolbar will be updated to include various additional functionality as listed below:

SQL Results toolbar

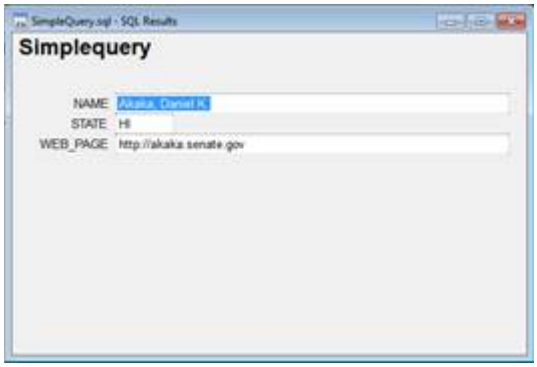
Starting from the left:

- **New –** this gives you a dropdown to pick or create a "new" something
- **Open –** this will open and load a file
- **Save –** this will save a file
- **Print –** this will print the active file
- **Cut –** this is the cut for the selected editor for the clipboard
- **Copy –** this is the copy for the selected editor for the clipboard
- **Paste –** this is the paste for the selected editor for the clipboard
- **Execute –** run and display the results of the SQL statement
- **Design –** this will return the product to the Design surface, the Execute and Design can toggle

- **Grid layout –** this is the default layout for the SQL Results:



- **Columnar layout –** this will change the results to a columnar layout:



- **Form layout –** this will put the SQL results into a form layout:

- **Add Row –** this will add a row at the bottom of the result set, and can be executed using the keyboard shortcut {CTRL-A}
- **Delete Row –** this will delete the active row in the result set
- **Save Row –** this will save or post the updates back to the underlying database through the respective data-access (ADO or BDE) and it can also be executed using the keyboard shortcut {CTRL-S}
- **Abandon Row –** this will abandon any changes made

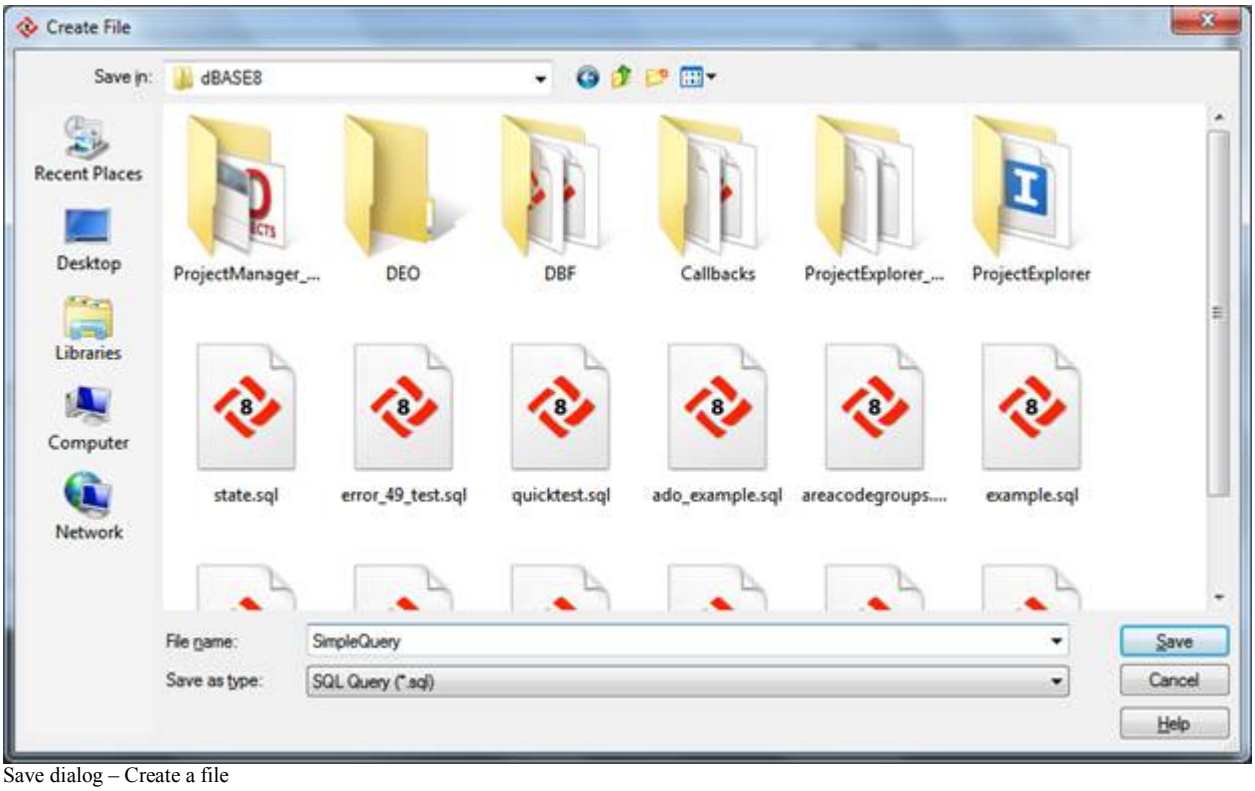
- **Find Rows –** This will return a set of rows and can be implemented using the keyboard shortcut {CTRL-F}

- **Sort Ascending –** this will put the results in a sort order from a – z
- **Sort Descending –** this will put the results in a sort order from z – a
- **First Row –** This will move the cursor to the first record in the result set. It can also be activated by using the keyboard shortcut {CTRL-HOME}
- **Previous Row –** This will move the cursor to the prior record in the result set
- **Next Row –** This will move the cursor to the next record in the result set
- **Last Row –** This will move the cursor to the last record in the result set. It can also be activated by using the keyboard shortcut {CTRL-END}
- **Context Sensitive Help –** this will take you to the help associated with the particular in focus part. For example, if you are on the Select statement, the the Help will start on the Select statement outline

The "Execute" process gives you full control over the look and presentation of the data, plus it allows you the ability to manipulate the dataset at time of execution.

Saving an SQL statement

In SQL Builder, before any SQL statements can be "Executed" they must be Saved. This can be accomplished by clicking the Save toolbar button or by click the File|Save or File|Save As… main menu, menu items. This will display the standard save dialog as shown below:

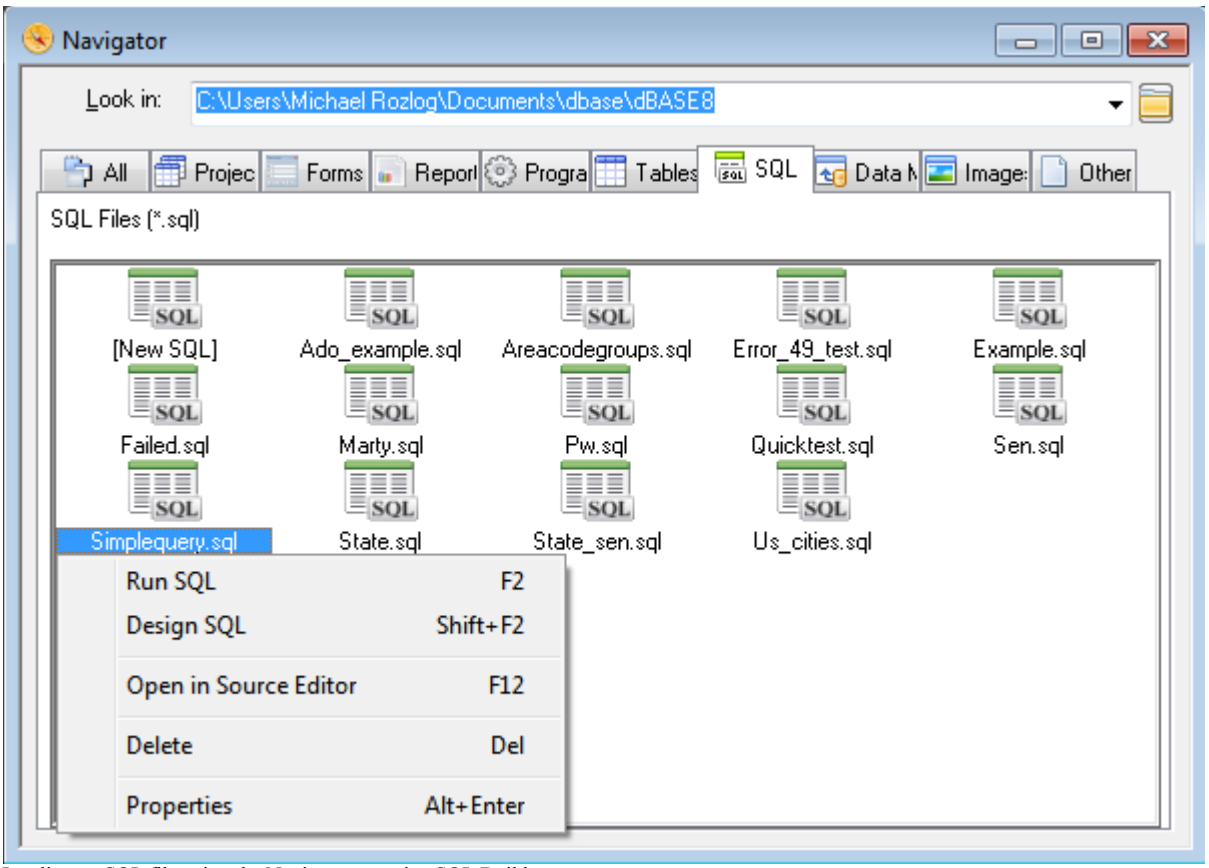


Save dialog – Create a file

Once the file has been named and pointed to the proper location, you can click the Save button to finish the save operation.

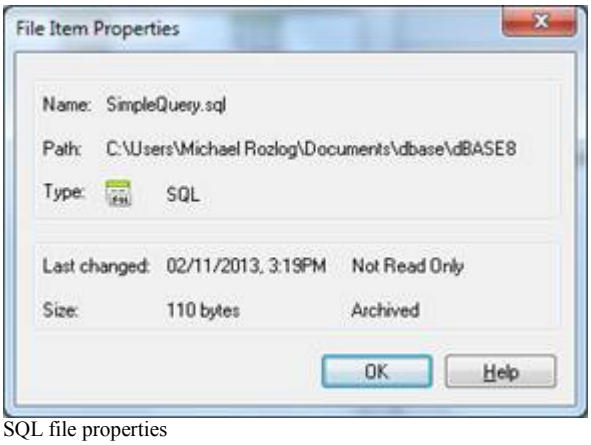
Loading an existing SQL statement

SQL Builder can be activated by loading a file with SQL inside that has a file extension of .sql, or by finding the specific SQL file in the Navigator as shown below:



Loading an SQL file using the Navigator to active SQL Builder

Using the Navigator gives the most options when loading an SQL file. Right-mouse clicks on the particular SQL file to be presented with the above options. You can Run SQL {F2} key, or open the SQL Builder designer using the Design SQL {Shift+F2} key, or go right to the source F12 key, or Delete {Del} Key the SQL file. Finally, they could get the SQL File properties {Alt+Enter}, as shown below:



SQL file properties

It will show the name and location of the file. When finished with the review, press the OK button to continue.

Joining Tables

SQL Builder fully supports Joins, and it automatically understands how to create Inner Joins with little fuss. The join type that is created by default is INNER JOIN, which is fine in most cases. However, for those servers that have no support of a JOIN clause, SQL Builder adds this condition to the WHERE part of the query, which is the case for dBASE (.dbf and .db) files. There are some additional rules that must be followed when it comes to JOINs in dBASE:

- All joins are left-to-right outer joins.
- All join are equi-joins.
- All join conditions are satisfied by indexes.
- Output ordering is not defined.
- The query contains no elements listed above that would prevent single-table updatability.

Conversely, for a little more information on Joins, here is an excellent explanation of Inner / Outer Joins types:

Assuming you're joining on columns with no duplicates, which is by far the most common case:

- An inner join of A and B gives the result of A intersect B, i.e. the inner part of a venn diagram intersection.
- An outer join of A and B gives the results of A union B, i.e. the outer parts of a venn diagram union.

Examples

Suppose you have two Tables, with a single column each, and data as follows:

A	B
1	3
2	4
3	5
4	6

Note that (1,2) are unique to A, (3,4) are common, and (5,6) are unique to B.

Inner join
An inner join using either of the equivalent queries gives the intersection of the two tables, i.e. the two rows they have in common.

- select* from a INNER JOIN b on a.a = b.b;
- select a.*,b.* from a,b where a.a = b.b;

a	B
3	3
4	4

Left outer join

A left outer join will give all rows in A, plus any common rows in B.

- select * from a LEFT OUTER JOIN b on a.a = b.b;
- select a.*,b.* from a,b where a.a = b.b(+);

a	B
1	null
2	null

3	3
4	4

Full outer join

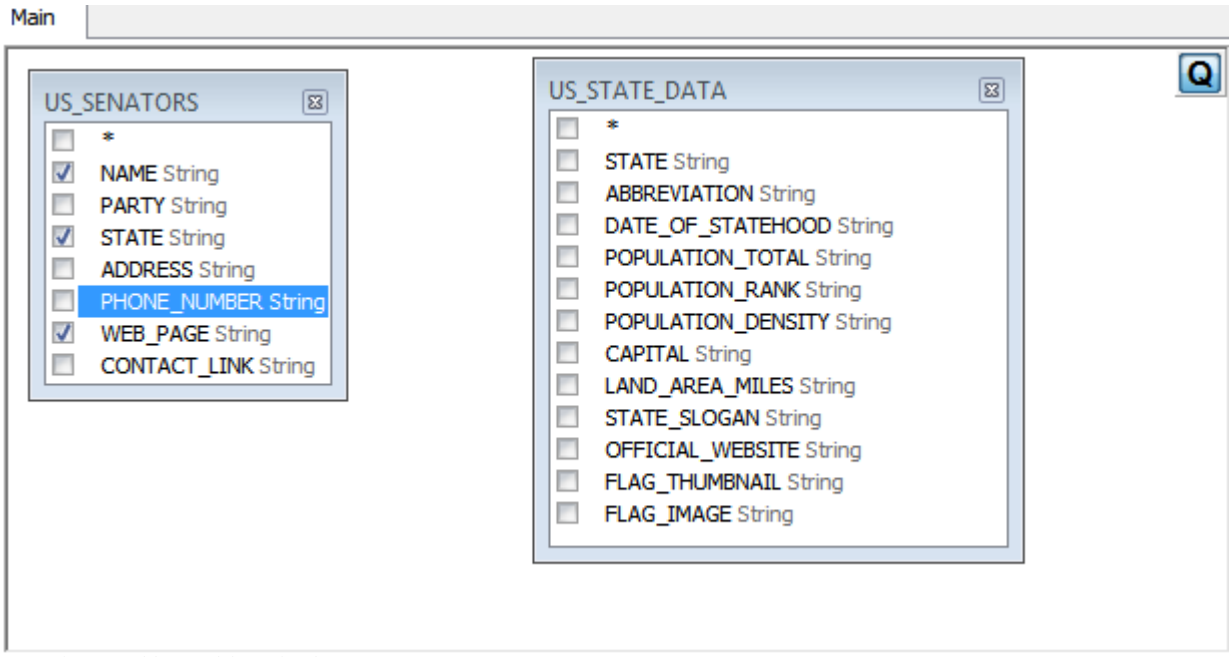
A full outer join will give you the union of A and B, i.e. All the rows in A and all the rows in B. If something in A doesn't have a corresponding datum in B, then the B portion is null, and vice versa.

- select * from a FULL OUTER JOIN b on a.a = b.b;

a	B
1	null
2	null
3	3
4	4
null	6
null	5

Taken from an excellent post and response from Mark Harrison from StackOverflow.com here: http://stackoverflow.com/questions/38549/difference-between-inner-and-outer-join (Above format modified for docs)

Our example will include two tables that have related information. Table A (US_SENATORS) and Table B (US_STATE_DATA) are listed below for reference:



Two reference tables used for a simple JOIN

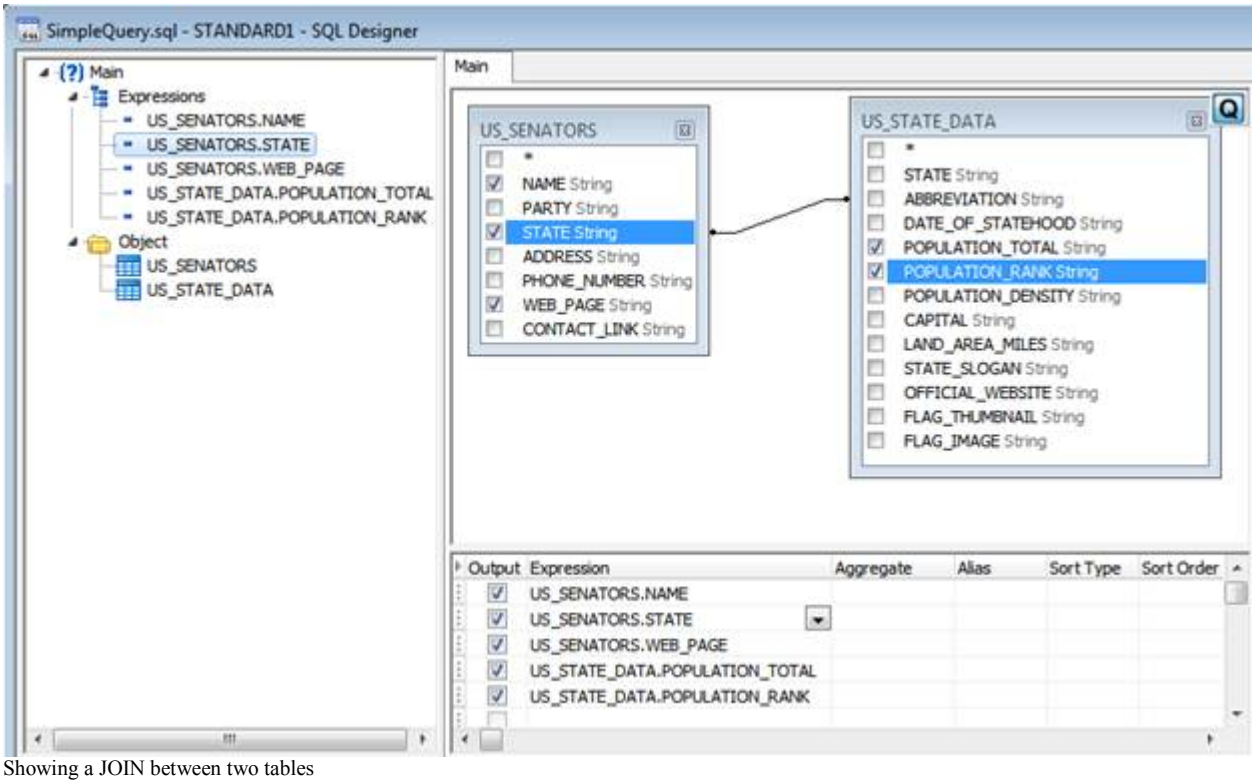
For this example, the Result set should include:

- US_SENATORS – Name
- US_SENATORS – State
- US_SENATORS – Web_Page

- US_STATE_DATA – Population_Total
- US_STATE_DATA – Population_Rank

Before the JOIN can be created, SQL Builder needs to have included fields marked. This means that Population_total and Population_rank need to be checkmarked. The US_SENATORS – State field is a 2 character abbreviation. That means the linkage between the two tables will be using that field and the US_STATE_DATA – Abbreviation field.

To create a link between two objects (i.e. join them) manually, you select the field you want to link and drag it to the corresponding field of the other object. After you finish dragging, a line connecting the linked fields will appear. Key cardinality symbols are placed at the ends of link when the corresponding relationship exists in the database. The results are shown below:



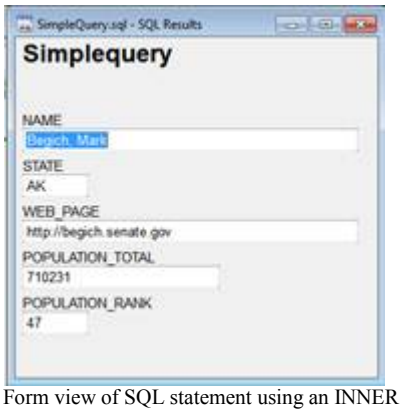
Showing a JOIN between two tables

In the above screen shot, notice that in the Query Tree pane (Left) that the US_SENATORS.STATE is selected and the same field in the Columns Pane (bottom) is selected, along with the drop-down arrow on the field as well. This is a quick way to go from field to field.

The code that was generated for this particular SQL looks like the following:

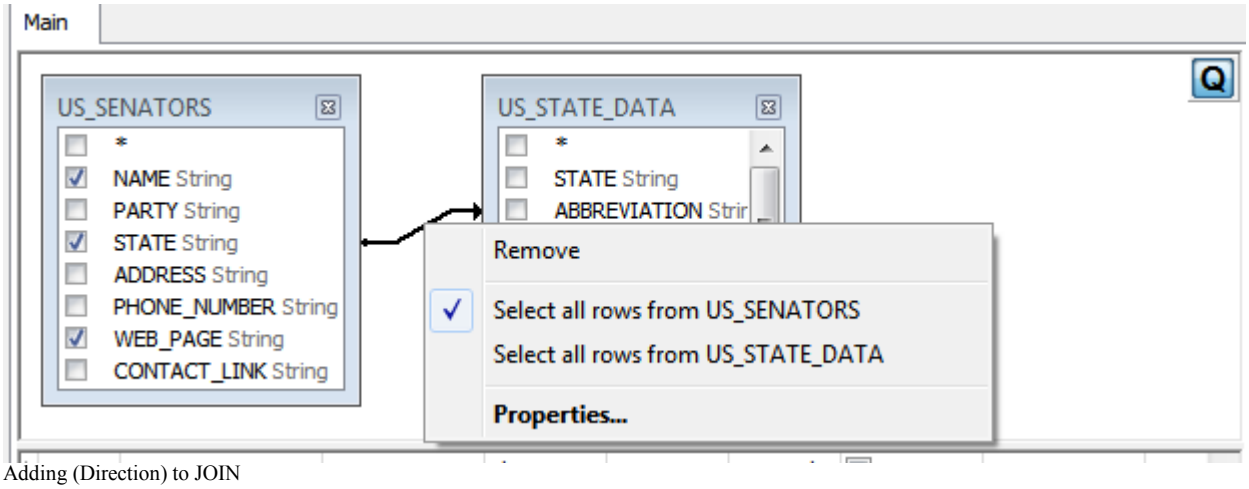
```
Select US_SENATORS.NAME,  US_SENATORS.STATE,   US_SENATORS.WEB_PAGE,
        US_STATE_DATA.POPULATION_TOTAL,
        US_STATE_DATA.POPULATION_RANK
From US_SENATORS
      Inner Join US_STATE_DATA On US_SENATORS.STATE =
                                          US_STATE_DATA.ABBREVIATION
```

When the Query is run, the Form view of the query looks like the following:



Form view of SQL statement using an INNER JOIN

To define join the type and other link properties, you can right click the link and select the Properties item from the context popup menu or double-click it to open the Link Properties dialog.



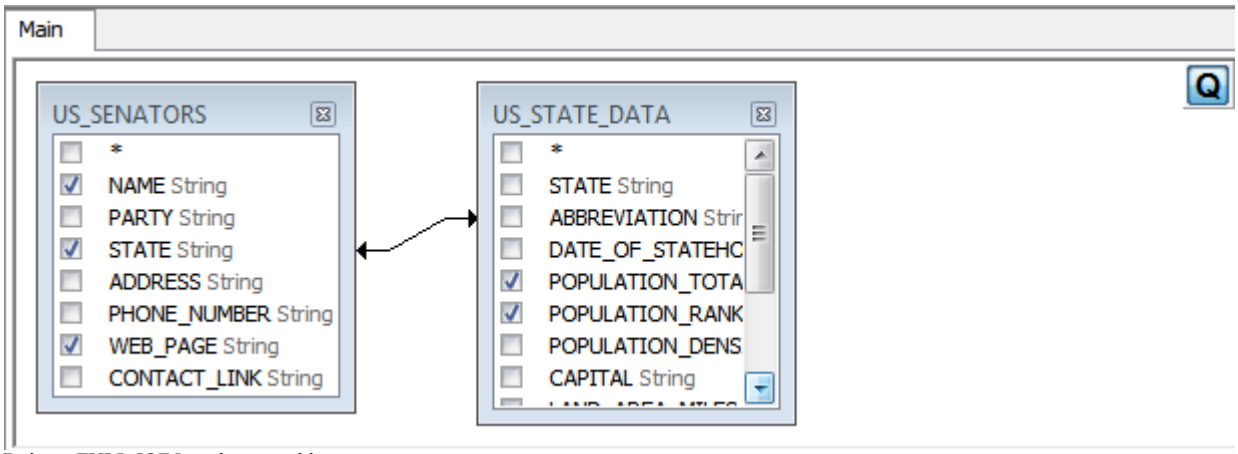
Adding (Direction) to JOIN

Notice in the above dialog, that the direction is now going Left on the JOIN and the SQL generated will be reflected below:

```
Select US_SENATORS.NAME,  US_SENATORS.STATE,  US_SENATORS.WEB_PAGE,
        US_STATE_DATA.POPULATION_TOTAL,
        US_STATE_DATA.POPULATION_RANK
from US_SENATORS
      Left Join US_STATE_DATA On US_SENATORS.STATE =
                                          US_STATE_DATA.ABBREVIATION
```

In this simple example, the output looks the exact same as doing a simple INNER JOIN.

There are times when you want to do a FULL JOIN and this is accomplished by making the linkage between the tables look like:



Doing a FULL JOIN on the two tables

The above source code would look like the following:

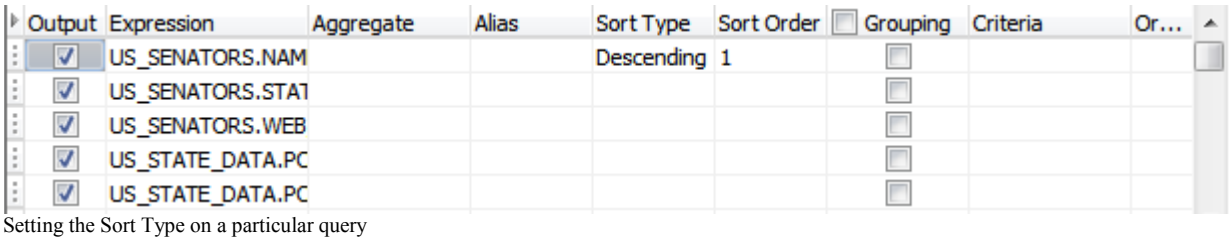
```
Select US_SENATORS.NAME, US_SENATORS.STATE, US_SENATORS.WEB_PAGE,
        US_STATE_DATA.POPULATION_TOTAL,
        US_STATE_DATA.POPULATION_RANK
From US_SENATORS
      Full Join US_STATE_DATA On US_SENATORS.STATE =
                                          US_STATE_DATA.ABBREVIATION
```

This can also be accomplished by right-mouse clicking on the linkage and selecting the Properties menu item:

Notice that both the Left and Right are selected and that represents a FULL JOIN. The outcome of this SQL is still the same as the Simple INNER JOIN that was first created, however the need to show the code generation difference was needed.

Sorting

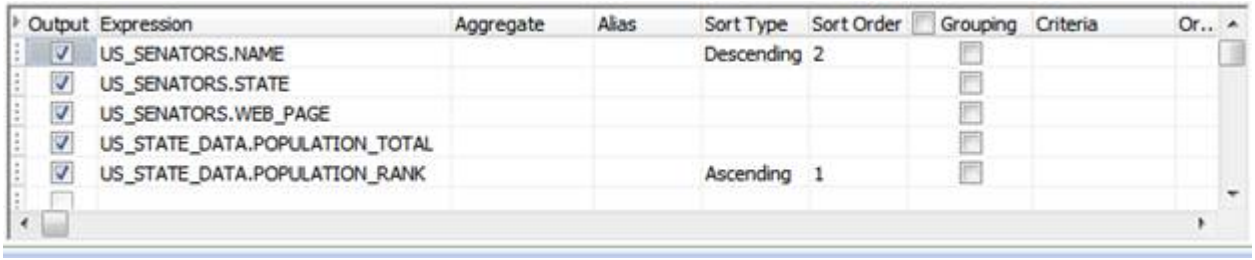
To define the sorting of the result dataset, you can use the Sort Type and Sort Order columns of the Columns Pane. Using the same query that was used in the JOIN section of the documentation, set the SORT Type on the US_SENATOR – Name in a descending order as shown below:



Setting the Sort Type on a particular query

This will result in the following:

The Sort Order column allows you to setup the order in which the fields will be sorted, in case more than one field will be sorted. To disable sorting, clear the Sort Type column for this field. Now add Sort field and change the type and order, as shown below:



Setting Sort Type and Order for different results

Now the results are based on the Population.Rank, highest first, and then the SENATORS – Name, which result in the following:

Initial result

Next record in line

Defining Criteria

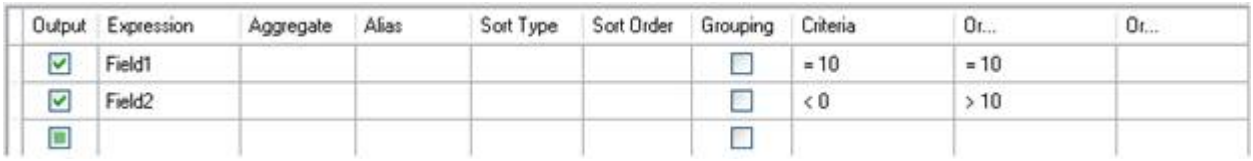
To define the criteria, you can use the Criteria and all of the Or columns of the Columns Pane (bottom).

In these cells, you should write the conditions omitting the expression itself. For example, to get the following criteria in your query:

WHERE (Field1 >= 10) AND (Field1 <= 20)

you should type ">= 10 AND <= 20" in Criteria cell of a Field1 expression.

Criteria placed in the Or columns will be grouped by columns using the AND operator and then concatenated in the WHERE (or HAVING) clause using the OR operator. For example, this visual representation will produce the following SQL statement. Please note that the criteria for Field1 are placed in both the Criteria and Or columns.



Output	Expression	Aggregate	Alias	Sort Type	Sort Order	Grouping	Criteria	Or...	Or...
☑	Field1					☐	= 10	= 10	
☑	Field2					☐	< 0	> 10	
■						☐			

Showing criteria creation

WHERE (Field1= 10) AND ((Field2 < 0) OR (Field2 > 10))

Some expressions may be of the Boolean type, for example the EXISTS clause. In this case you should type "= True" in the Criteria column of such expressions or "= False" if you want to place a NOT operator before the expression.

Note: the most common practice to learn for how to build queries with criteria is to write it once by hand and see how it will be parsed and represented visually.

Grouping

To build a query with grouping, you mark the expressions for grouping with the Grouping checkbox.

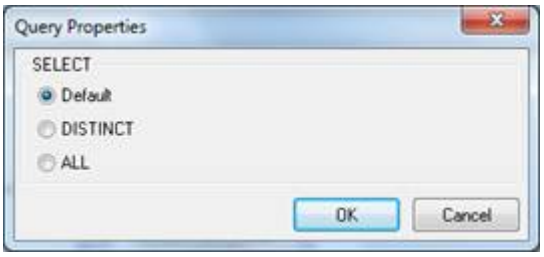
A query with grouping must have only grouping or aggregate expressions in the SELECT list. Thus, SQL Builder allows you to set the Output checkbox for grouping and aggregate expressions. If you try to set this checkbox for a column without the Grouping or Aggregate function set, a Grouping checkbox will be set automatically to maintain the validity of the result SQL query.

When the Columns Pane (bottom) contains columns marked with the Grouping checkbox, a new column called "Criteria for" appears in the grid. This column specifies the appliance of criteria to the expression groups or to their values.

For example, you have a column "Quantity" with the Aggregate function "Avg" in your query and you type the "> 10" in the Criteria column. Having the "for groups" value set in the Criteria for column, the result query will contain only groups with an average quantity greater than 10, and your query will have the "Avg(Quantity) > 10" condition in the HAVING clause. Having the "for values" value set in the "Criteria for" column, the result query will calculate the Average aggregate function only for records with a Quantity value greater than 10, and your query will have the "Quantity > 10" condition in the WHERE clause.

Query Properties

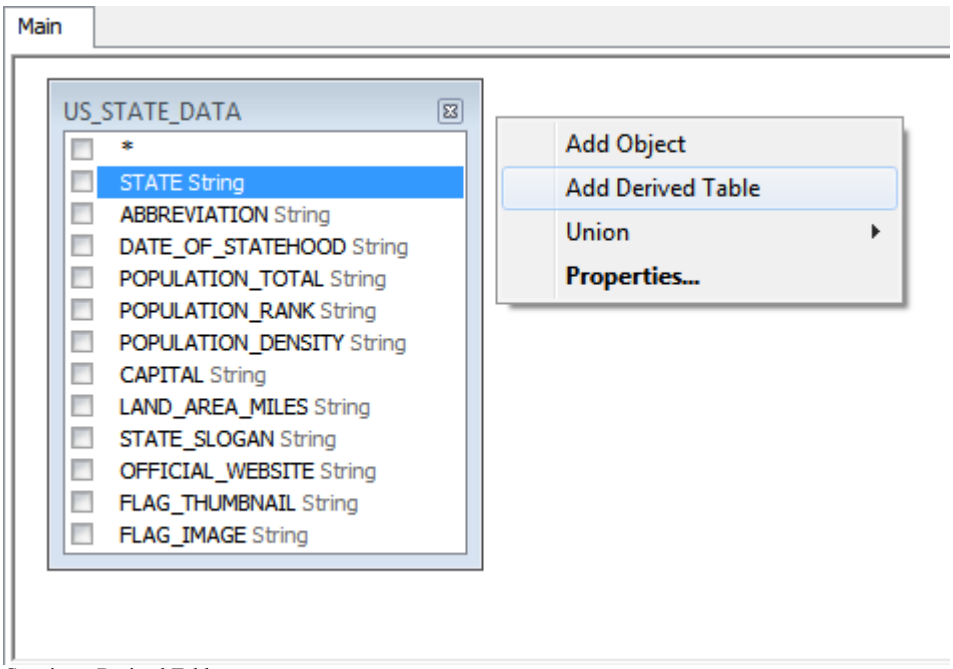
Various database server specific options are managed within the Query Properties dialog. You can open it using the context popup menu of the Query Building Area.



Derived Tables

A Derived tble is a sub-query used as a datasource for the main query.

To add a derived table, you should right click the Query Building Area and select the Add Derived Table item from the context popup menu.

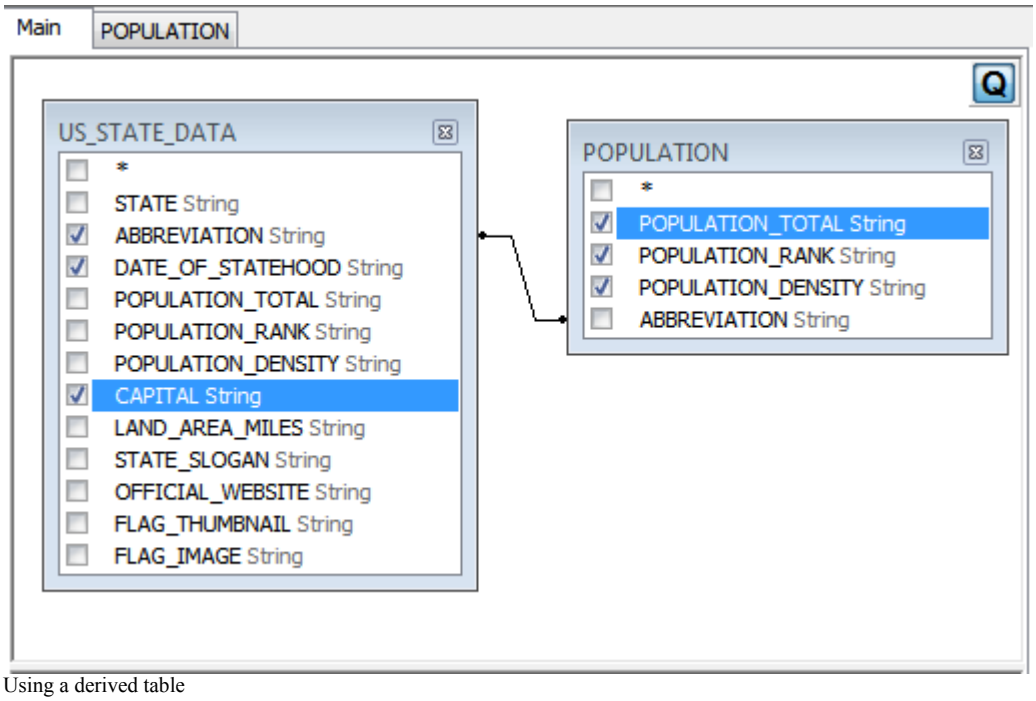


Creating a Derived Table

A new object representing the newly created derived table will be added to the query building area, and the corresponding tab will be created for it. This tab allows you to build it visually in the same way as the main query. Another way to switch to the corresponding derived table tab is to right click the caption of an object representing the derived table and select the "Switch to derived table" item from the context popup menu.

You can set an alias for the derived table the same way as for an ordinary database object.

You can always go back to the main query and switch to any sub-query or derived table using tabs above the Query Building Area (middle) or using the Query Structure Tree (Left).



Using a derived table

Unions

SQL Builder fully supports Unions. UNION combines the results of two or more queries into a single result set that includes all the rows that belong to all queries in the union.

Union sub-queries are managed within the Union Panel in the top-right corner of the Query Building Area. Initially there is only one union sub-query labeled with the "Q" letter. All required operations are performed by means of context popup menus.

Union sub-queries can be grouped with other sub-queries and joined with different operators (UNION, UNION ALL, EXCEPT, INTERSECT).

- To add a new union sub-query, select the New Union sub-query menu item.
- To enclose the sub-query in brackets, select the Enclose in Brackets menu item.
- To move the sub-query or bracket to the top of the query (the topmost sub-query is the left one), select the Move Left menu item.
- To move the sub-query or bracket to the bottom of the query, select the Move Right menu item.
- To remove the sub-query or bracket, select the Remove menu item.
- To change the union joining operator, select the necessary operator in the list of supported operators in the context popup menu.

Chapter 11 Report Designer

Chapter

11

Designing reports

Reports provide non-editable views of data for formatted print or screen output. You can create reports to answer questions that may involve elaborate queries across a range of databases. A report can focus and manipulate data in many useful ways.

This group of topics shows you how to

- Use the Report wizard to automatically generate reports (using the wizard is the recommended way to begin creating a report)
- Understand the Report designer structure and objects
- Modify a report, changing its appearance and functionality
- Perform aggregate calculations
- Use multiple streamFrames that point to the same or different rowsets
- Create a variety of specialty labels by using the Label wizard

Report wizard

You can quickly create useful reports by using the Report wizard. You specify which table or query contains the data you want to display in the report, and the wizard links to it automatically. The rest of the wizard's options let you do the following:

- Display detail rows or just summary information.
- Specify fields to be included in the report.
- Group the report by specific fields. You can nest subgroups within groups.
- Choose aggregate operations that can be applied both to a group and to the entire report.
- Specify layout style, including a drill-down option, which displays summary information at the top of the page and details farther down.

- Specify a report title.
- Include the date.
- Include page numbers, which causes the report to display one screen full at a time.

The Report wizard does so much that for many reports you won't need to go any further. You can, however, add complex query statements to your reports by writing code or using the SQL designer to generate SQL statements. And you can add advanced reporting capabilities, as needed, in the Report designer.

It's easiest to begin creating a report by using the Report wizard. You can then modify the design in the Report designer. By using code, you can add a great deal of analysis to your reports and provide more sophisticated and useful pictures of the data in one or more tables.

To use the Report wizard

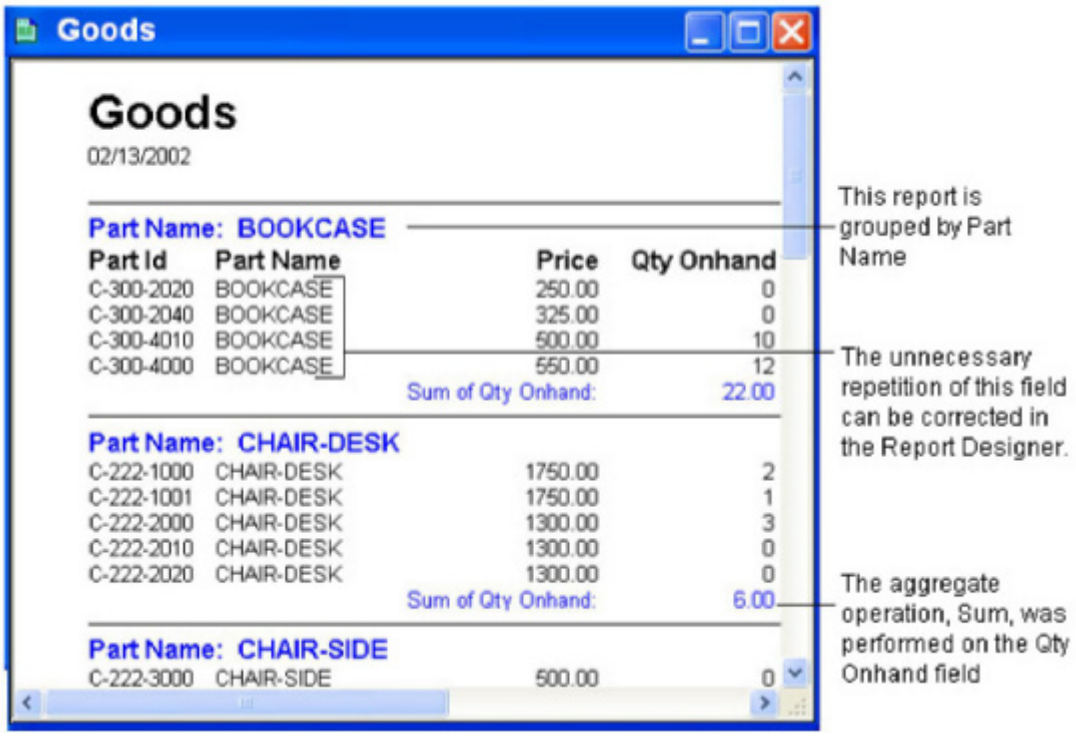
1. Choose File | New | Report. Or, double-click the leftmost Untitled icon on the Reports page of the Navigator. The New Report dialog box appears.
2. Click the Wizard button.

For help on any wizard page, click the Help button on that page.

Example of a report created with the Report wizard

This example uses a GOODS.DBF table that might be used by a Purchasing Department to track current inventory levels. The report answers these questions: What are the total quantities on hand of each furniture type, and what is the cost per unit. Here is the final report:

Figure 0.1 Wizard-generated report on a GOODS table



The report displays a heading for each furniture type, listing the individual styles below each heading. Grouping by furniture type lets you do subtotals, and several other calculations, for each type.

Aggregate operations analyze the values of a selected summary field within a group or over the entire report. You can use any field in a table, even if it is not included in the report. Also, you do not have to specify a field for grouping to use it as a summary field.

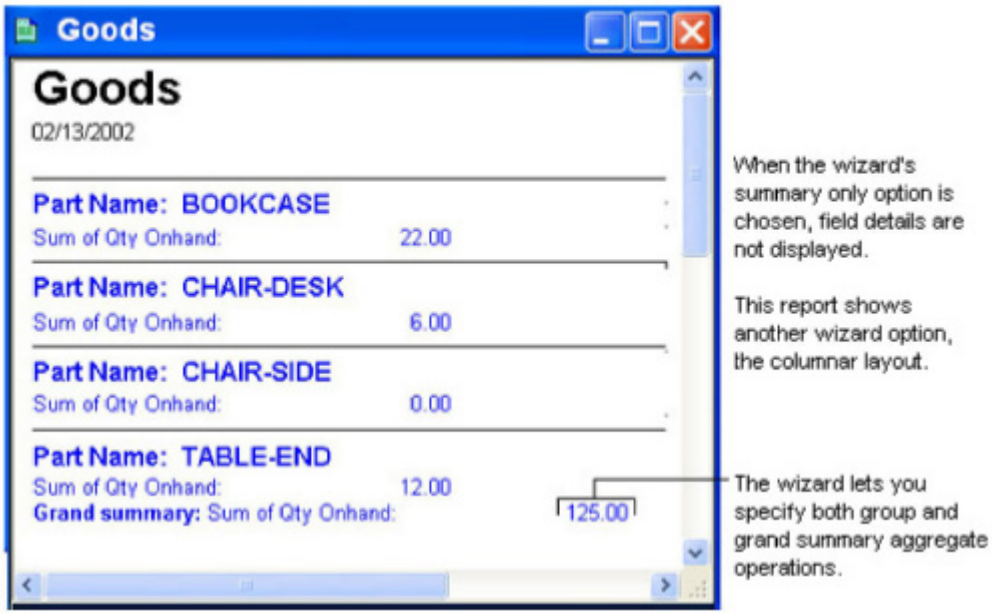
The Qty Onhand field was selected from step 5 of the wizard, and Sum was selected as the Aggregate Operation. This totaled the values in the Qty Onhand field for each grouping of rows, so that a total of the Qty Onhand column appears in each Part Name group.

Because the report is grouped by Part Name, the Part Name column is redundant. To delete a column, see “Deleting columns (fields) from a report”

Wizard-generated Summary Report

This variation of a wizard-generated report on GOODS.DBF, is the most direct way of answering the question: What is our inventory of each furniture type and what is our total inventory for all units. The finished report looks like

Figure 0.1 Wizard-generated Summary Report

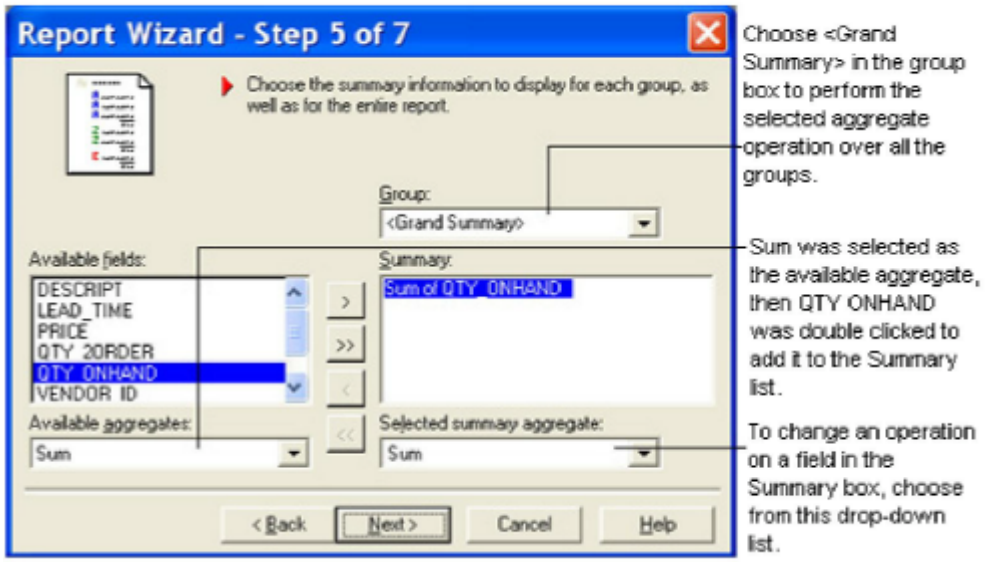


In the wizard, the Qty Onhand field was specified as the summary field for the Part Name group. Sum was chosen as the aggregate operation for this field.

To create the grand total, <Grand Summary> was chosen from the Group drop-down list in step 5. Qty Onhand was selected in the Available Fields box, Sum was chosen from the Available Aggregates list, and the right arrow (>) button was clicked to add the Qty Onhand field to the Summary box, as shown below. The Qty Onhand field’s associated aggregate operation then appears in the Selected Summary Aggregate box

.

Figure 0.1 Adding grand total in the Report wizard



Report designer elements

The Report designer provides a visual design surface where you can modify reports created with the Report wizard and build new reports from scratch.

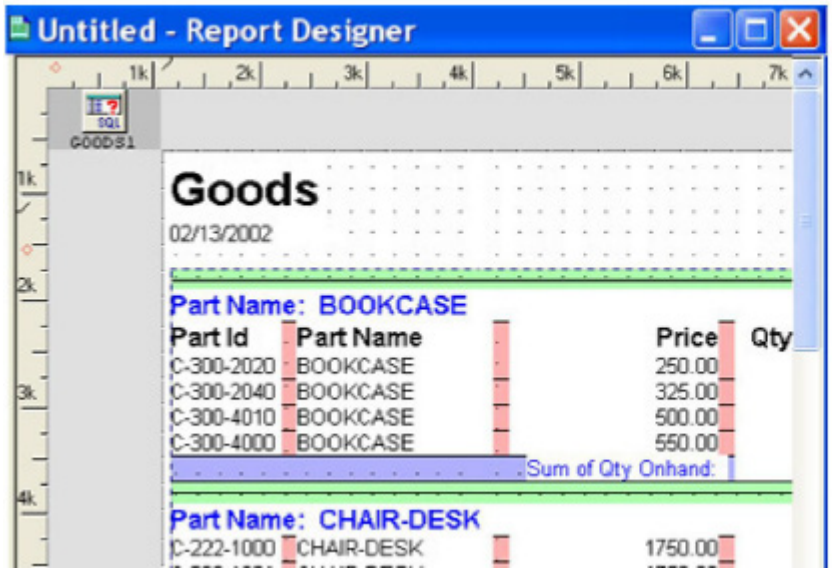
The Report and Group panes

You can view the report in two panes:

- The Report pane, where you visually design the report
- The Group pane, which displays the hierarchy of the groups in the report.

When you first open the Report designer, the split bar between the panes appears at the far left edge of the Report designer window. To open the Group pane, drag the split bar to the right..

Figure 0.1 Report in Design mode with Group view displayed



The Group pane shows the hierarchy of objects in the report.

- The dotted-line frame labeled Pagetemplate1 is the report object that determines the appearance of the page, such as the background color. Here is where you would place a report title. A report may include more than one pageTemplate object, so that different pages can have different layouts.

  When creating a report, you place data access components on the pageTemplate object.

- The inner dotted-line frame with the vertical label Streamframe1 is a streamFrame object. This object displays rowset data that is streamed from linked tables (specified in its *streamSource* property). One or more streamFrame objects may be contained within the pageTemplate object. Whatever is placed in the streamFrame area of a report will be displayed when the report is printed.

  If you're using standard user interface components, place them on the streamFrame.

- The Group1-Headerband displays the label of the grouped field. Groups are contained in a streamSource object and rendered in a streamFrame object.

- The detailBands are the streamFrame objects that contain the rowsets streamed from the linked table. Each detailBand contains data from an individual row in the table.

The Report pane shows the report appearance with the corresponding structures shown in the Group pane marked.

- The outer dotted area represents the margins of the actual report page (the pageTemplate object).
- The inner dotted line represents the data rows of the report (the streamFrame object).
- The individual fields of the row are columns in the report (the detailBand objects).

Modifying report in the Report designer

This section illustrates how to use the Report designer to modify the design and adjust the appearance of a report.

Deleting columns (fields) from a report

To delete a column,

1. Click in the column beneath the column heading to select the column, and press Delete.
2. Click the column heading itself, and press Delete. The remaining columns stay where they are.

If you make a mistake and delete the wrong object, choose Edit | Undo Delete.

Adding columns (fields) to a report

Once you have created a live Query object on a report (usually by dragging the desired table icon to the designer surface), the Field palette is populated with live fields linked to the table data. When you drag a field from the palette to the design surface, a detailBand is created to display the field as a column in the report. The column is linked to the table's field by a codeblock in the *text* property of the detailBand's Text object.

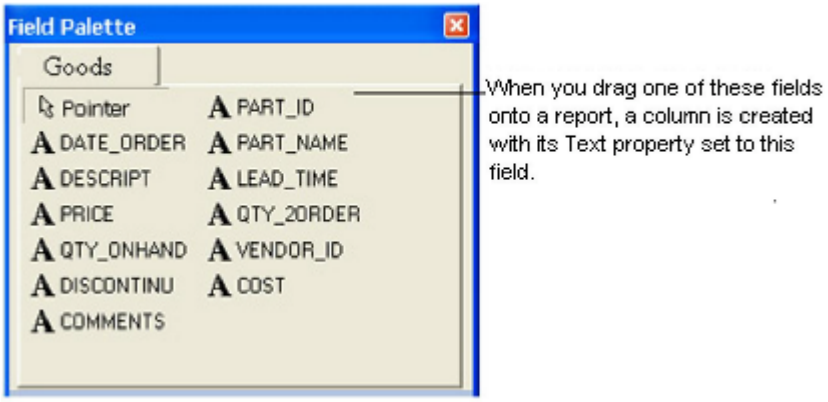
To add a column,

1. Make sure the report has a Query object whose *active* property is set to *true* and that returns a rowset from the desired table.
2. Display the Field palette (View | Tool Windows, and check Field Palette–Report/Label Designer). The palette shows the active fields of the linked table as Text components
3. Drag the desired Field component from the palette to the report.

When you drag a live Field component to a report, the field's name is automatically added as a column heading. To specify placement (or omission) of the automatic column label, choose Tool Windows | Customize Tool Windows and specify your preference on the Field Palettes page.

.

Figure 0.1 Field Palette containing active fields



If you need to move the added column, remember to reposition its heading, as well.

Suppressing duplicate field values

To suppress duplicate field values, set the *suppressIfDuplicate* property of a detailBand's Text object to *true*.

Displaying default values in a blank report field

To display a default value in a report to substitute for blank values in a field,

1. Determine the field in which you want to display the default value, and select that field's Text object in the Inspector (streamSource1.detailBand.<text object>). This example uses the NAME field from the query object, query1.
2. Enter an appropriate codeblock into the *text* property of the Text object (click the tool button in the property box). For example, the following code displays No Value for every blank value in the field:

```
{||this.form.query1.rowset.fields["NAME"].value == "" ? "No Value":
this.form.query1.rowset.fields["NAME"].value}
```

Adding a floating dollar sign to field values in reports

To add a floating dollar sign to the values in a field,

1. Select the field (represented on the report by a Text object).
2. In the Inspector, select the Text object's *picture* property under the Edit category.
3. Click the tool button to display the Template Property Builder.
4. Select the Numeric page.
5. Choose the @$ symbol from the Template Symbols box.

You can also do this by using the Code Block Builder to create the code.

Adding page numbers

To include a page number on each page of a report, drag the PageNumber component (from the Component palette's Custom page) to the report page, positioning it where you want the number to appear. (This is a custom component installed with the dBASE PLUS samples.)

Creating a page number from scratch

1. Place a Text component on the report's page (on the pageTemplate object).
2. Open the Inspector and make sure that the newly added Text component has focus.
3. Locate the text property in the Inspector, and click on the type selection drop-down button (Shaped like a down arrow with the letter "T" on it) and choose Codeblock from the drop-down list.
4. Click on the wrench tool for the text property to open the Code Block Builder.
5. Delete the default text including the quotes and enter the following:

```
this.parent.parent.reportPage
```

Make sure the Expression radio button is selected, and press the OK button to close the dialog

Drill-down reports

Drill-down reports display summary information for a report at the beginning of the report. The details of the report appear toward the bottom. Hence the name *drill-down*—users can drill down from the summary at the top to the details at the bottom.

The Report wizard offers you the option to create a drill-down report. You can also create a drill-down report in the Report designer.

Controlling drill-down reports in the Report designer

In the Report designer (or if you print the report), the summary information of the report is in the headerBand and footerBand of the reportGroup class. The details of the report are in the detailBand.

In a non-drill-down report, the bands are rendered in this order (this example groups on STATE):

1. headerBand for report's reportGroup
2. headerBand for state of "CA"
3. detailBand 1 for state of "CA"
4. detailBand 2 for state of "CA"
5. detailBand n for state of "CA"
6. footerBand for state of "CA"
7. headerBand for state of "PA"
8. detailBand 1 for state of "PA"

9. detailBand n for state of "PA"
10. footerBand for state of "PA"
11. footerBand for report's reportGroup

However, in the drill-down report, the bands are rendered in this order:

1. headerBand for report's reportGroup
2. footerBand for report's reportGroup
3. headerBand for state of "CA"
4. footerBand for state of "CA"
5. headerBand for state of "PA"
6. footerBand for state of "PA"
7. detailBand 1 for state of "CA"
8. detailBand 2 for state of "CA"
9. detailBand n for state of "CA"
10. detailBand 1 for state of "PA"
11. detailBand n for state of "PA"

...and so on

The drillDown property

You can control the way the drill-down feature works by setting the drillDown property on the reportGroup class. This property is an enumerated type, with the following possible values:

Table 11.1 Values for the *drillDown* property

Value	What happened
0 None (not a drill-down report)	
1 Drilldown (standard drill-down report)	All the headers and footers are rendered first, and then the details
2 Drilldown (repeat header)	The same as 1, but the headers are rendered again with the details
3 Drilldown (repeat footer)	The same as 1, but the footers are rendered again with the details
4 Drilldown (repeat header and footer)	The same as 1, but the footers and headers are rendered again with the details

Adding standard components to a report

By adding components from the Report's component palette, you can extend a report's functionality so that it can do some of the same things a form can do.

Important

When working with components in Reports, keep these points in mind:

- Report components may be placed only on a pageTemplate, not on a band.
- When you copy and paste a component, its object hierarchy may vary in the following ways:
  - Its parent will be the parent of the currently selected component; or,
  - If the currently selected object can hold a component (a pageTemplate), then that object will be the parent; or,
  - If nothing is selected, then the pageTemplate will be the parent.
- Mouse events of a component whose parent is a band are not available.
- All components available for use on reports have *canRender*( ) and *onRender*( ) events.

- Components on a pageTemplate can be referenced directly as well as through an elements array (as in forms).
- If you do not want a report component to be printed, set the component's *printable* property to *false*.

Changing the report's appearance

You can place data from any source in streamFrames that can be sized and positioned anywhere on the report page. You can create a variety of borders around streamFrames, labels, or fields. You can control all aspects of the appearance of text. And you can specify colors for the entire report's background, for text, and for streamFrames.

Creating report borders

dBASE PLUS offers a choice of several different styles of borders. You can create borders around all report fields and columns and labels at once or around individual objects to set off things like grand totals. You can also place borders around streamframes (giving groups a boxed appearance).

To set borders around each column in the report, set the report's form.PageTemplate1 *gridLineWidth* property to a positive number. (*gridLineWidth* is in the Inspector's Miscellaneous category.)

To create a border around an individual object, including a streamframe, select the object and set its *borderStyle* property to one of the styles in the drop-down list beside the property. (*borderStyle* is in the Inspector's Visual category.)

For no border, set *borderStyle* to zero.

Setting background color in reports

To set a report's background color,

1. Select the report's PageTemplate object in the Inspector (form.PageTemplate1).
2. Click the wrench tool beside the *colorNormal* property to display the Color Property Builder.
3. Select a color, or create your own, and choose OK.

Setting background image in reports

To set a report's background image,

1. Select the report's PageTemplate object in the Inspector (form.PageTemplate1).
2. Click the wrench tool beside the *background* property to display the Image Property Builder.
3. Click the wrench tool beside the Image box of this dialog box to browse for an image file. As soon as you select it, you can see the background image in the Report designer.

Performing aggregate (summary) calculations

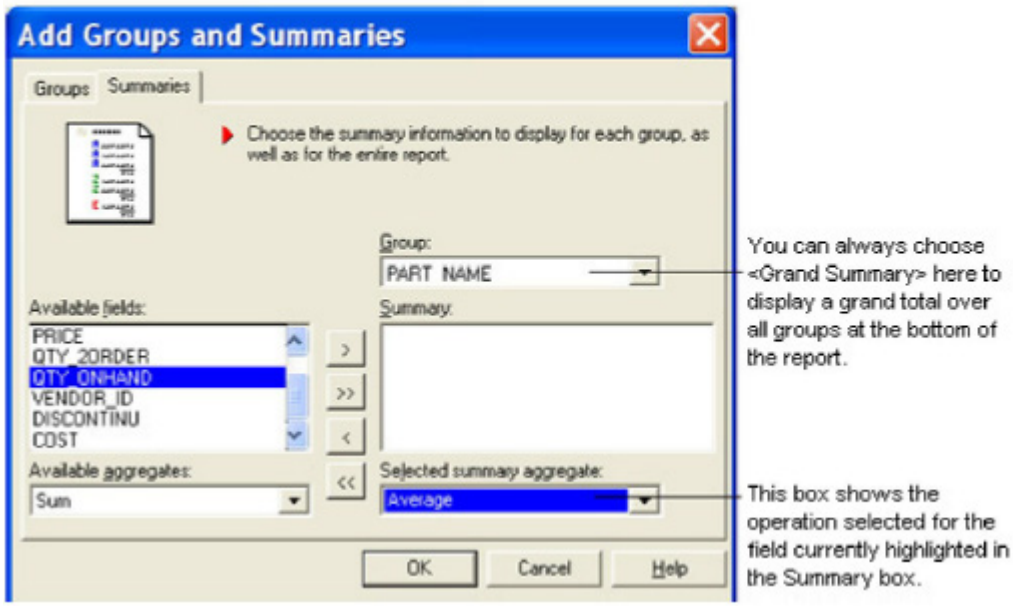
You can perform aggregate calculations (sum, minimum, maximum, count, average, standard deviation, variance) on fields within groups in a report and for the report as a whole.

To do aggregate calculations you must first specify a group of fields on which to perform the summary operation (unless you are summarizing data for the entire report).

To perform an aggregate operation,

1. Choose Layout | Add Groups And Summaries
2. On the Group page of the dialog box, select the field or fields on which you want to group the data. If you are going to do only a grand summary over the entire report (such as a grand total), then skip this step.
3. Click the Summaries tab of the dialog box.

Figure 0.1 Aggregate calculation on a Report



4. From the Group drop-down list, select the group on which you want to perform an aggregate operation. (Grand Summary is always available.)
5. From the Available Fields list, select the field on which you want to perform the operation.
6. From the Available Aggregates drop-down list, select an operation.
7. Click the arrow button to move the field and its operation to the Summary list.
8. Repeat for as many fields as you want.

If you want to change the aggregate summary operation for a field already in the Summary box, select that field in the Summary box and use the drop-down list below it to select a new operation.

If you need very sophisticated calculations beyond the operations offered in the Available Aggregates box, you could also create summary calculations in fields by writing methods for their events. Select the object linked to the field, and type your code into one of its report-specific events, such as

- *onDesignOpen*, activated on opening a report
- *preRender*, activated before a report runs
- *canRender*, activated before a component is rendered, to determine whether the component will be displayed.

Designing a report with multiple streamFrames

The streamFrame object makes it possible to create a rectangular area of any size or position to display data rows from a linked rowset (which is set in the streamFrame's *streamSource's* rowset property).

For example, the Label wizard adds several streamFrames to a report, with their common *streamSource* pointing to the same rowset of customer addresses. By sizing each streamFrame to match sheets of adhesive labels, the wizard can create a report that prints a set of mailing labels.

To add a second streamFrame to an existing report,

1. From the Component palette, Report page, drag the streamFrame component to the design surface of the report.
2. The streamFrame object appears closed, as a box. You can drag this box diagonally to any desired size.

In addition, by using the Source editor, you can create additional streamFrames and set each streamFrame's *streamSource* property to point to different rowsets. For example, you could add two streamFrame objects side-by-side on a report, with one frame displaying sales representatives grouped by city and the adjacent frame displaying prospective customers grouped by city.

Creating printed labels

The easiest way to create mailing labels and many other types of printed labels is by using the Label wizard. Choose File | New | Labels or double-click the Label icon on the Reports page of the Navigator, then choose Wizard. After you select the table or query file you want to use, the wizard does the following:

- Sets up a common address format for you, or you can create your own format
- Sorts the labels on a field you specify
- Sets up the page for the type of label sheets you have (many choices)
- Lets you create a calculated field

Labels are a type of report, given the extension .LAB. Label files you create are listed on the Reports page of the Navigator.

If you choose to modify labels after using the wizard, or if you design them from scratch, you'll be working in the Label designer. The Label designer is similar to the Report designer, except that you have additional choices on the menu.

Chapter 12 Designing tables

Chapter

12

Introduction to designing tables in dBASE PLUS

The foundation of any application is the system of tables that store the data. dBASE PLUS gives you powerful tools for creating and managing databases, whether your application needs a simple table or complete access to an enterprise client/server database system.

This section is a conceptual introduction to designing and creating tables:

- Table terms and concepts
- Table design guidelines
- Table structure concepts

Terms and concepts

You should know these essential terms:

- An ***application*** is a complete system of tables and related forms, queries, reports and other components that handles a data management need.
- A ***database*** is a collection of one or more *tables* that store and classify information, plus related files such as index, graphic, and memo files. Each dBASE table in a database is a distinct file with a .DBF extension.
- A ***table*** consists of one or more horizontal ***rows*** (sometimes known as records) that contain information about a specific person, place, or thing.
- Each row contains one or more ***fields***. A field contains one category of information, such as a person's name, a phone number, or an invoice date.
- A ***field type*** describes the kind of information stored in the field; for example, date, character, logical, numeric.

When you first create the table, you choose a *table type.* Your choices are standard tables (.DBF and .DB) and other supported databases for which you have configured a BDE alias. Then you define each field's name, field type, width, and decimal (if a numeric or float field). You can also create an index on the field, which lets you arrange rows in a useful order.

The dBASE PLUS interface adjusts automatically to accommodate the type of table you are working with. For example, if the table you are working with supports data entry constraints, you can specify them in the Inspector while designing a Table. Otherwise, data entry constraints are unavailable in the Table designer.

.

Figure 0.1 Components of a Table

Table design guidelines

The following sections illustrate typical design issues using a hypothetical small business, a shop that sells diving equipment.

Identifying the information to store

To develop a system of tables for an application, begin by identifying all of the *relevant* information you need to manage—unnecessary data wastes disk space and distracts users from the task at hand.

You might start by looking at the order form you use day to day. Write down all the information you think you need, without attempting to organize it yet, as shown in the following example:

- Products ordered
- Customer name, address, phone number, credit standing
- Order number
- Shipping information, including when it was shipped
- Products purchased
- Purchase date and time
- Salesperson taking the order
- Customer signature
- Special notes about the customer

Next, review this list to see what's really relevant and what you can do without. For example, the name of the salesperson taking the order might not be important, or you might decide you don't need to track the exact time of purchase.

Classifying information

After you identify the information you need, review the list and begin to classify the information into distinct groups. Identify the separate entities (such as persons, places, and things) and activities (such as events, transactions, and other occurrences). In general, each table should contain only one kind of entity or activity. The fields in each table identify the attributes of that entity or activity.

Reviewing the list in the previous topic reveals two separate entities (customers and products) and one separate activity (orders). Each of these components has unique attributes. When you reorganize the list according to these categories, you might come up with a new list similar to the following one:

- Customers, including customer name, address, city, state, postal code, phone number, credit standing, signature, notes, and so on
- Orders, including order date, order number, sales date, amount paid, and so on
- Products ordered, including product name, sales price, quantity ordered, and so on

Determining relationships among tables

To better define your tables, you need to determine how they relate to each other.

Single versus multiple tables

Each table should have a specific purpose. It's often better to create several small tables and link them together, rather than try to store everything in one large table. Keeping everything in a single large table usually forces you to store redundant data.

For example, if you stored the complete customer information with each order, you would be entering the customer's name and address with every order. Not only does this procedure invite errors, but it makes it difficult to update information if something as simple as a customer's phone number should change. In addition, redundant data wastes disk space.

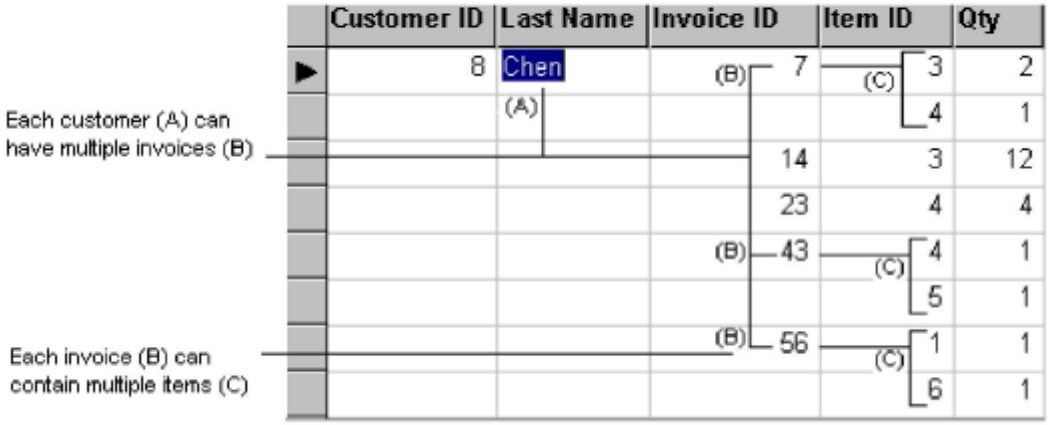
Use multiple tables to minimize the amount of data that appears more than once. By storing the name and address information once in a Customer table, you have only one location to update if that information changes. When a customer places an order, the order information goes in a separate table that can be linked to the name and address table.

In our sample case, three distinct tables have emerged to contain data: one for customers, another for orders, and one for the items ordered. We can call the tables Customer, Orders, and Lineitem.

One-to-one and one-to-many relationships

With multiple tables in an application, it's important to understand the relationships among entities and activities. In each relationship, is there a one-to-one correspondence, or does an entry in one correspond to many entries in another? Or is there no direct relationship?

Figure 0.1 One-to-many relationships



For example, a customer can place several orders, and an order can contain one or more items. These are *one-to-many* relationships. Each order is associated with one customer, a *one-to-one* relationship.

The query might then relate three tables:

- From the Customer table comes the customer name, NAME
- From the Orders table comes the order number ORDER_NO
- From the Lineitem table comes the stock number STOCK_NO and selling price, SELL_PRICE.

The query results show each customer name followed by many orders, and under the orders, a list of the items and prices in the order (Figure 12.2).

Parent and child tables

When you relate two tables in a query, form, report, or data module, you establish one as the parent and the other as the child table. As you select a row from the parent table, you see the corresponding child row or rows.

Linking tables in a parent-child relationship lets you easily find rows in the child table. For example, you can set up the Customer table as the parent, and Orders as the child. Then, when you move to a new row in the Customer table, the row pointer in the Orders table moves to the orders for that customer automatically. Similarly, the Orders table becomes the parent to the Lineitem table, so that selecting an order also selects the items in the order.

The parent and child tables are linked on a common field, called the linking field. In our example, a query links the Customer and Orders tables on the CUSTOMER_N (customer number) field. In dBASE PLUS, the linking field in the child table must have an index. As the example shows, a single table can be both parent and child in the same query.

Minimizing redundancy

Using multiple tables reduces redundant information. In addition, don’t store information you can easily calculate, unless the calculation requires excessive processing effort or you need an audit trail. For example, if you were creating tables for our hypothetical dive shop, you might want to store the total invoice amount, but not the total sales tax, which dBASE PLUS can easily calculate.

In general, indexed fields that are used for linking should be the only fields that contain redundant information in related tables. Identical data in index fields is necessary for linking tables. In this example, it is the way to identify the same customer in both tables.

Choosing index fields

Indexes make it easier and faster to process information in a table. With multiple tables, indexes are also necessary to link related tables together. Most tables should have at least one index, to organize rows and link to related tables, but too many indexes can slow performance.

To identify which fields to index, ask the following questions:

- What will users know when they search for information? For example, in the dive shop tables, users might want to search for a customer name, customer number, order number, or order date. Consider indexing on these fields.
- What are the common threads that tie the information together? For example, a customer number could be a common field between the Orders table and the Customer table, and the order number could be a common field between the Orders table and the Lineitem table..

Defining individual fields

For each field, you define its name, type, size, decimals (if a numeric or float field), and index (optional). The specifics of field types are discussed in the following section. Here are some overall guidelines:

- Use one piece of information per field. For example, put city, state, zip code, and country data into separate fields, because you might want to process the information in each field separately. However, do not split certain information, such as street number and street name, unless you need to process rows by street name or street number separately.
- Keep field sizes to a minimum, without being excessively restrictive, to conserve disk space. If you intend to total a numeric field, you must define a field large enough to hold the total, not just individual values.
- For indexed fields, use abbreviated codes instead of long character fields wherever possible. For example, instead of duplicating the entire customer name in every order, use a short customer code to simplify data entry, indexing, and linking. This results in more efficient indexes and makes it easier to update information.
- Define fields in a logical order in the table. The order you define is the default way in which users will see the table. In general, put indexed fields toward the beginning of the table, and put similar information together in a sensible sequence.
- Use descriptive, unique field names. Be consistent when naming fields that contain similar data. Standardize field names shared across tables if possible (this is not permitted with some SQL databases).

Table structure concepts

This section provides a general overview of basic table structure, with specific reference to the dBASE 7 table type.

Table names

See your database software documentation to determine valid file names for its tables. For example, an Access table has no extension requirement because it is stored within an Access database with an .MDB extension. On the other hand,.DB is the required extension for Paradox tables and .DBF for dBASE tables.

The table name should indicate its purpose and be easy to remember. For example, if a table contains employee information, you might call it EMPLOYEE.DBF.

Table types

The *table type* determines the file format of a table.

The table type you define depends on the way you plan to use the table. If you expect to use the table only with dBASE PLUS applications, the dBASE Level 7 format is recommended for its flexibility and rich feature set. If the table is to be shared with other applications, consider the most useful format for all applications involved.

The dBASE Level 7 format offers all the features of the previous dBASE file formats, including expression indexes and extensive table-, row-, and field-level security.

The dBASE PLUS interface adjusts automatically to accommodate the type of table you are using. For example, if the table with which you are working supports it, you can specify data-entry constraints in the Inspector while working in the Table designer. Otherwise, data-entry constraints are unavailable in the Table designer.

Field types

Each field has a defined *field type*, which determines the kind of information it can store. For example, a character field accepts all printable characters including spaces. You can define up to 1,024 fields in a table.

A dBASE (.DBF) table can contain the following field types.

Table 12.1 dBASE field types for level 7 tables

Field type	Default size	Maximum size	Index allowed?	Allowable values
Character	10 characters	254 characters	Yes	All keyboard characters
Numeric	10 digits, 0 decimal	20 digits	Yes	Positive or negative numbers
Float	10 digits, 0 decimal	20 digits	Yes	Positive or negative numbers. Identical to Numeric; maintained for compatibility.
Long	4 bytes	N/A	Yes	Signed 32 bit integer, range approximately +/-2 billion. Optimized for speed.
Double	8 bytes	N/A	Yes	Positive or negative number. Optimized for speed.
AutoIncrement	4 bytes	N/A	Yes	Contains long integer values in a read-only (non-editable) field, beginning with the number 1 and automatically incrementing up to approximately 2 billion. Deleting a row does not change the field values of other rows. Be aware that adding an autoincrement field will pack the table.
Date	8 bytes	N/A	Yes	Any date from AD 1 to AD 9999
TimeStamp	8 bytes	N/A	Yes	Date/Time stamp, including the Date format plus hours, minutes, and seconds, such as HH:MM:SS
Logical	1 byte	N/A	No	True (T, t), false (F, f), yes (Y, y), and no (N, n)
Memo	10 bytes	N/A	No	Usually just text, but all keyboard characters; can contain binary data (but using binary field is preferred)
Binary	10 bytes	N/A	No	Binary files (sound and image data, for example)
OLE	10 bytes	N/A	No	OLE objects from other Windows applications

The field type determines what you can do with the information in the field. For example, you can perform mathematical calculations on values in a numeric field, but not on values in a logical field.

The field type also determines how the data appears in the field. For example, a date field, by default, displays dates in the MM/DD/YY format (such as 02/14/96). The display of field data is also affected by the settings of the Windows control panel and the settings defined by using the BDE Administrator.

Other table types, such as SQL tables, may have different field types. Refer to your server documentation for specific details.

Chapter 13 Creating Tables

Chapter

13

Creating tables

This chapter describes the dBASE PLUS Table wizard, designer, and other tools for designing table structures. Here you will find procedures for creating structures, indexes, and performing other database design tasks. It assumes you are familiar with the basics of table design presented in Chapter 12, “Introduction to designing tables in dBASE Plus”, and covers the following topics:

- Supported table types
- Using the Table wizard
- Using the Table designer
- User-interface in the Table designer
- Restructuring tables (overview)
- Creating custom field attributes
- Specifying data constraints
- Creating and maintaining indexes
- Referential integrity

Supported table types

dBASE PLUS provides a Table wizard and Table designer to quickly create tables in any supported table format. Although a particular database application may provide the fullest support for its native format, you can conveniently lay out the basic structure of its tables in the dBASE PLUS Table designer and view any table in Run mode.

- All table access is handled through the Borland Database Engine (BDE), which includes drivers to support the following table, and database formats.
- BDE-standard (no other software or BDE alias required):
  - dBASE

  - Paradox
- Other desktop database formats:
  - FoxPro 2.5
  - Microsoft Access 95/97

  The software application must be installed and running, with aliases assigned in the BDE Administrator. Alternatively, BDE's ODBC socket supports any ODBC database. For example, if Microsoft Access is not installed, you can connect to an Access database via ODBC. For details, see BDE Help (BDEADMIN.HLP).
- SQL enterprise client/server database formats:
  - Oracle
  - Sybase
  - Informix
  - Microsoft SQL Server
  - IBM DB/2
  - InterBase

  The database server system must be installed and running, with aliases assigned in the BDE Administrator.

A BDE alias is a short name used as a shortcut to a client/server database or to a directory containing database files. BDE aliases are required for Access, Foxpro, and all SQL client/server systems. You may also use BDE aliases for dBASE and Paradox tables for convenience or application portability, although it is not required.

Although you can create tables in any supported format, this section shows how to use the Table wizard and designer to quickly create tables in the dBASE Level 7 table format, which is the most feature-rich and convenient. Some of the dialog boxes and capabilities might not apply to a particular database you are using. Please see the documentation for your database for guidance on implementing tables in its native format.

Note

The terms "database" and "table" are often confused. A database consists of a set of files, including indexes, memo, and graphics files, and one or more tables that may be related by key fields. A table consists of an ordered set of rows (records), each row containing a set of defined data fields. The larger, client/server database management systems are considered more database-oriented. The smaller "desktop" database applications are sometimes said to be table-oriented, although when related tables and index files are stored in a directory, that directory may be considered a database.

Using the Table wizard

To use the Table Wizard to create a new table,

1. If you intend to create a table type other than dBASE or Paradox, click the Tables tab in the Navigator, and select an alias from the Look In drop-down list. Your new table will be of that alias's database type.
2. Choose File | New | Table (or double-click the Untitled icon on the Tables page of the Navigator). The New Table dialog box appears.
3. In the New Table dialog box, choose Wizard.
4. In step 1 of the wizard, select the fields you want from the available tables.
5. Click Next and select the table type.
6. Choose Run Table to save the table
7. Choose Design Table to continue the design process
8. In step 2, select a table format type. If you have selected a database alias in the Navigator as described in step 1 of this procedure, that alias appears as the default table type and cannot be modified. To change it, you must exit the wizard and choose a new BDE alias from the Look In box on the Tables page of the Navigator.

Click the Help button on any step of the wizard for details about the options available with that step.

Using the Table designer

This section outlines a short procedure for creating tables in the Table designer. To create a new table by using the Table designer,

1. Choose File | New | Table.
2. In the New Table dialog box, choose Design.
3. The Table designer appears showing a default template for the first field.
4. Set the table type in the Inspector. You can select a BDE-standard table type or the table types of those databases for which you have created BDE aliases.
5. In the Table designer, type a name (no spaces for dBASE files) in the Name field. (You have to name a field before you can specify any of its other attributes.)
6. Specify values for the remaining attributes (Type, Width, Decimal, Index) by typing what you want, or by selecting a value from the drop-down list, or by clicking the spinbox arrows. You can tab through these to select the default values.

To create additional fields, press Return when you have finished specifying a field. Or press the Down arrow key. Or right-click and choose Add Field from the context menu.

You can generate new fields in rapid succession by naming each, then pressing the Down arrow key. After naming all the fields in your table design, you can go back and set or reset the attributes for each field.

Warning!

Later on in your work, do not use functions such as CHR(_), LTRIM(_), RTRIM(_), TRIM(_), or IIF(_) that vary the field width in the key expression.

Table designer tips

- To add, insert, or delete fields, right-click in the Table designer window to display a context menu, and choose the appropriate command.
- To reorder the sequence of fields, place the insertion point in the field number box—it becomes a hand—and move the field to the desired position in the list.
- For information on .DBF field types see, Table 12.1, “dBASE field types for level 7 tables,”
- For more information on elements of the Table designer, see the next section.

User- interface elements in the Table designer

This section provides a detailed description of the user interface elements and common tasks of the Table designer.

To open the Table designer to modify an existing table, right-click the table on the Tables page of the Navigator, and choose Design Table from the context menu, or select the table and click the Table Design button on the toolbar.

- The Table designer lists the fields defined in the table, along with the attributes for each field.
- **Field** contains a number that identifies the field in the table. Field numbers are consecutive, automatic, and read-only. They determine the default order in which fields appear in the Table window.
- **Name** is the name of the field (up to 31 characters for dBASE PLUS). You can enter letters, numbers, and underscores, but no other characters. The first character must be a letter. Paradox and most SQL tables allow spaces; dBASE tables do not.

Note

Do not use reserved words for the field name. For example, DATE.

- **Type** is the field type. Select the type you want from the list. The type you select determines what kind of data the field will contain. It also determines whether you can set the width, decimals, and index options for this field.
- **Width** is the field size. In the case of dBASE tables you can change field size for character, numeric, and float fields only (all others have fixed width). Never use functions that vary field widths.
- **Decimal** is the number of digits allowed to the right of the decimal point (for float and numeric fields only). In the case of dBASE tables, float and numeric fields, by default, have no decimals selected. You can set decimals to a maximum of 2 less than the width value you define. The total width must be 20 characters or less. This includes decimal settings, the decimal point, and an optional minus sign.
- **Index** determines whether to index rows using the values in this field (you can set an index on character, date, float, and numeric fields in dBASE tables). Select Ascend to index this field in ascending order (for character fields, this is ASCII order, or the order determined by your language driver). Selecting Descend indexes this field in descending order, and None (the default) omits this field from indexing (or removes an existing index associated with this field).

  If you select Ascend or Descend for a dBASE table, the Table designer creates an index for the field in the multiple index file (.MDX) associated with the table.

  To set a primary key on a dBASE 7 or Paradox table, choose Structure | Define Primary Key. Some SQL types also support this.

You can also set other field attributes or create custom field attributes by selecting the field and opening the Inspector. See Help.

Resizing columns

You can resize or move columns and move rows in the Table designer.

- To resize a column, point to the column border. When the pointer changes to a double-headed arrow, the column is outlined and you can drag the border until the column is the size you want.
- To move columns, point to the title of the column you want to move. When the pointer changes to a hand, the column is outlined and you can drag it to its new location.
- To set multiuser locks or default table type and other properties, choose Properties | Desktop Properties | Table tab.

Note

If you want to see rows that have been marked for deletion when the table is in Run mode, the Deleted option must be unchecked in the Desktop Properties dialog box. The rows will appear and the work deleted will appear in the status bar for those records marked for deletion. There is not a "delete flag" for each record.

Getting around in the Table designer

In the Table designer, each horizontal row of properties represents one field (or column) in the table you are designing or modifying. To add, change, or delete data, first select the field by clicking with the mouse or by using keyboard shortcuts.

To go to a specific field number,

1. Choose Structure | Go To Field Number or press Ctrl+G.
2. Type the number of the field to go to and click OK.

Adding and inserting fields

You can add a new field to the table by either adding a row at the end of the fields list or by inserting a row anywhere in the list.

To add a new field to the end of the fields list, choose Structure | Add Field (or right-click anywhere in the Table designer and choose Add Field from the context menu).

To insert a new field between other fields, select a field, and choose Structure | Insert Field, or right-click and choose Insert Field from the context menu. The new field's row of properties appears above the one you selected.

Moving fields

To move a field, changing its order in a table, point to the field number in the leftmost column. When the pointer changes to a hand, drag the row up or down to its new location.

Deleting fields

To delete fields from a table,

1. Click anywhere in the property row of the field you want to delete.
2. Choose Structure | Delete Current Field (or right-click and choose Delete Current Field from the context menu).

The Table designer deletes the field definition. If the table contains rows, the data in this field is deleted as soon as you save the table structure.

Note

Short-cut keystrokes can also be used to add (Ctrl+A), insert (Ctrl+N) and delete (Ctrl+U) fields.

Saving the table structure

Save the table design to keep the structure you've created. If you haven't yet saved a new table design, doing so creates the table and any associated files (such as .DBT and .MDX files in the case of dBASE tables).

To save changes to a table design, do one of the following:

- If it's a new table, choose File | Save.
- To save an existing table under a new name, Choose File | Save As.

If you are saving for the first time, or if you choose Save As, the Save Table dialog box appears.

If you are saving an older dBASE .DBF file, it will be saved in the new dBASE Level 7 file format with all the extended capabilities built in.

Note

The table will no longer be usable in lower versions of dBASE.

Type a valid file name. Choose a destination drive, directory, and database, if needed, and then choose OK. dBASE PLUS creates or updates the table and any associated files.

Note

You may not use a file-name extension ending in a "T".

Abandoning changes

Abandon changes to a table design if you want to cancel creating a new table or discard the changes you have made to an existing table.

To abandon changes,

1. Choose File | Close to close the Table designer.
2. Choose No when asked to save changes.

Restructuring tables (overview)

It's easy to change the structure of a table, even if the table contains row data.

If the table is empty, you can make any valid changes you want to the table structure except change the table type. If the table contains rows, however, you need to be more careful about the changes you make—and you should make a backup copy of the table before attempting to change its structure.

When you change the structure of a table, the Table designer makes a backup copy of the old table, creates a new table with the revised design, and attempts to copy all the data from the backup table to the new table. However, each time you change the structure of this table, the backup copy that the Table designer created is overwritten. That is why you should make your own backup copy with a unique name or in another directory.

This section assumes you are using BDE-standard table types (dBASE or Paradox). You can also change the structure of the table types of other databases connected via BDE aliases. For information on restructuring these tables, see the documentation of the respective manufacturer.

Important guidelines for restructuring

When you change the structure of a table, the Table designer uses the field name and field position to determine how to transfer information to the new structure.

Warning!

If it cannot find a corresponding field in the new table, the Table designer *does not copy the data from the fields in the backup table*; instead, the information is lost when the backup table is deleted.

To prevent losing data that you want to keep, save the table structure frequently as you make changes and confirm that they are completed successfully.

If you change the type of a field, the Table designer does its best to convert data to the new type. Some conversions are relatively straightforward, such as converting date, logical or numeric fields to character. However, radical conversions (such as a memo field to a date field) might produce results you don't want. In addition, the Table designer does not copy data that is invalid in the new field type. For example, attempting to copy the value "123ABC" from a character field to a numeric field fails because letters aren't valid entries in numeric fields.

In addition to these guidelines, remember that if you delete a field in a table that contains rows, you lose the information in that field permanently. You can recover the information only if you have made a backup of the table.

Changing the structure

To change the structure of a table,

1. Open a table in Design mode
   If you are working in a shared environment, you see a prompt to open the table exclusively. Choose Open Exclusive to open the Table designer.
1. Make a working copy of the table (choose File | Save As and specify a new name for the table). The working copy now has focus.
2. Change the field definitions you want. You cannot change the table type.
3. When you finish, choose File | Save. In addition to saving your changes, the Table designer also copies associated files (such as .MDX and .DBT files).

Note

Open the restructured table in Run mode to verify that your data is in the condition you want. If not, you can revert to your original table if you worked from a copy.

Printing the table structure

To print the table structure for future reference,

1. Open the table in the Table designer.
2. Choose File | Print.
3. Choose the print options you want and click OK.

Table access passwords

In addition to restricting access to networks and servers, you can limit access to sensitive tables by setting passwords directly on those files. The dBASE file format provides extensive table-,row-, and field-level access restrictions. For more information on dBASE PLUS security features, see Chapter 15, “Setting up security”.

Creating custom field attributes

Custom field attributes specify how a field will be displayed in a form or report, irrespective of the form’s default control settings. You can create custom field attributes in the Table designer to control special properties and events on forms and reports. Whenever a field is *dataLink*ed to a control, all custom field attributes are copied to the control.

Custom field attributes are named field properties that contain a string value. These attributes form an active data dictionary that functions at both design and runtime. Attributes are listed under the Inspector’s Custom category.

Properties assigned to fields by creating custom field attributes are not streamed. Therefore, you can change the attribute in the table without having to change your code. If you later change the field’s attributes, the changes are automatically applied to the control *dataLink*ed to the field. No change to a report or form is necessary.

By using custom field attributes, you can cause a table’s field to have its own special font, color, or format that will be reproduced on any form or report whose controls are *datalink*ed to it. For example, you can assign a picture attribute to a PHONE field. When you *dataLink* an entryfield control to the PHONE field, that entryfield control will automatically take on the *picture* property that was assigned as the field’s picture attribute.

To create custom field attributes in the Table designer,

1. Select the field for which you want to create a custom attribute. Open in the Inspector.
2. Right-click and from the context menu, choose New Custom Field Property...
3. In the dialog box, enter a name for the new field attribute. For example, if your table contains a phone number field whose data you want to appear in forms in a particular phone number format, you would type picture to add the *picture* property (which provides data format templates, not images).
4. Now enter a value for the new field attribute. If you used the *picture* property, you might add a template value for the property, such as 999-999-9999 for a USA phone number template.
5. The new field attribute appears in the Inspector, listed under the Custom category.

To edit or delete custom field attributes,

1. In the Inspector, select the custom field attribute.
2. Right-click to display the context menu.
3. Choose Modify or Delete Custom Field Property from the context menu.

No checks are performed on the attribute name; be sure not to create attributes, such as "Name," that will cause undesired property name conflicts. Attributes with names not used by the component simply become custom properties of the component.

Specifying data-entry constraints

If supported by the database type, you may be able to specify data-entry constraints—rules that govern the values you can enter in a field. If you want to make sure that the values users enter in a field meet certain conditions, specify a data-entry constraint for that field.

You can specify data-entry constraints for each field in the Inspector when you create or modify a table that supports them, such as a dBASE or a Paradox table.

The Inspector displays different data-entry constraints depending on the field type.

Table 13.1 Data-entry constraints

Validity check	Meaning
Required	Every row in the table must have a value in this field.
Minimum	The values entered in this field must be equal to or greater than the minimum you specify here.
Maximum	The values entered in this field must be less than or equal to the maximum you specify here.
Default	The value you specify here is automatically entered in this field. You can replace it with another value.

Creating and maintaining indexes

Rows in dBASE tables can be organized either by indexing or by sorting. Both methods arrange rows in a specific order, but in completely different ways. Relational databases require index files; sorting creates a separate table with a different organization.

This section describes both indexing and sorting in a dBASE table. It covers the following topics:

- Indexing versus sorting
- Simple indexes and complex indexes
- Design concepts and guidelines for indexes
- Adding, modifying, and deleting indexes
- Sorting data to a separate table
- Creating indexes for Paradox tables

Note

The material in this section applies to dBASE, Paradox, and SQL indexes. However, specific guidelines and procedures might differ. If you're using SQL tables, see your database documentation.

Indexing versus sorting

Indexing and sorting are two approaches for establishing the order of data in a table. You use them to answer different needs in an application. In general, you index a table to establish a specific order of the rows, to help you locate and process information quickly. Indexing makes applications run more efficiently. Use sorting only when you want to create another table with a different natural order of rows.

Indexing orders rows in a specific sequence, usually in ascending or descending order on one field. Indexing creates a list of rows arranged in a logical order, such as by date or by name, and stores this list in a separate file called an *index file*. A dBASE index (.MDX) file can have up to 47 indexes, but only one controls the order of rows at any time. The index that is controlling the order is the current master index.

Note

dBASE PLUS stores indexes in multiple index (.MDX) files, and recognizes older .MDX files. You can design and maintain multiple indexes using the Manage Indexes dialog box.

Sorting creates an entirely separate copy of the current table with the rows in a different order. You're likely to use sorting infrequently, only when you want to create a separate table with a different natural order.

Here is a summary of key differences between indexing and sorting:

- **Creating tables.** Indexing creates an index file that consists of a list of rows in a logical row order, along with their corresponding physical position in the table. Sorting a table creates a separate table and fills it with data from the original table, in sorted order.
- **Arranging rows.** Both indexing and sorting arrange rows in a specified order. However, indexing changes only the logical order and leaves the natural order intact, while sorting changes the natural order of the rows in the new table.
- **Processing operations.** Certain operations are much faster using indexes, such as searching for data, running queries, and so on. Some operations, such as linking tables, require indexes.
- **Using functions.** With indexes, you can order rows using fields and dBASE PLUS methods. With sorting, you can use fields only, in ascending or descending order.
- **Adding rows.** If you add rows to an indexed table, the index is updated automatically so that the rows appear in the correct order. If you add or change rows in an already-sorted table, you might need to sort it again.
- **Mixing field types.** With indexing, you must convert field values to a common field type, for example, converting the sale date to a character type. With sorting, you can order rows on fields with different field types; for example, you can sort on customer number (a character field) and sale date (a date field), without converting them to a common field type.
- **Mixing order.** With indexes, the entire index is either ascending or descending. With sorting, you can mix fields sorted in ascending and descending order.

In general, use indexing to make processing more efficient in data entry forms, queries, and reports. The only significant costs are that index files require extra disk space, and processing time is required for ongoing automatic maintenance.

Sorting or exporting rows

Sorting a table copies its contents to a separate table and arranges rows in the order you specify in the new table.

Tip

In general, use sorting only when you want to export data to another application or table type. Sorting is useful whenever you want to create a separate table for reporting or other purposes. Use indexing instead when you want to make data entry, querying, and reporting tasks faster and more efficient.

When you sort, the *source table* is the table containing the rows you want to copy, and the *target table* is the new table (and new table type, if you want) to contain the copied rows. Sorting does not change the data in the source file.

When you sort a table, all fields in the source table appear in the target table. You select the fields on which to sort rows.

dBASE PLUS sorts data in case-sensitive alphabetic order, using the sort order specified by the language driver in the BDE Administrator. Sorting starts with the first character in the key and proceeds from left to right. Punctuation comes before numbers, numbers before letters, and uppercase letters before lowercase letters.

Note

Make sure you have enough available disk space to store the table on the target drive.

To create a sorted table or export table data to another table type,

1. Open the table you want to sort in Run mode.
2. Choose Table | Sort Rows to Table. The Sort Rows dialog box appears.
3. Specify a target table. This is the path name of the new-sorted file. Click the Target Table name tool button to display a Save dialog box. If you want to export the table data to another table type, choose the new table type in the box at the bottom of the Save As dialog box.
4. Select the field(s) on which to sort rows, and click the > button to move them to the Order By list. The order in which the selected fields appear in the Order By list determines the order of the sort. The target table contains all fields from the source table.
5. Select each Order By field, then specify the sort order.
6. When you have finished, click OK. dBASE PLUS creates a new table. If the target file exists, dBASE PLUS asks whether to overwrite it. The rows you selected are copied to the target table and sorted as you specified, starting with the first Order By field.

dBASE index concepts

Before you create indexes on dBASE (.DBF) tables, you need to be familiar with a few general concepts.

- **Multiple index (.MDX) files.** When you create an index, it is stored in a file with the file-name extension.MDX. Each index has a name (sometimes called a *tag*) that defines the index uniquely in the .MDX file.

  A table's main .MDX file is called the *production index* file. The production index file opens automatically when you open a table, so its indexes are automatically available—though no index sets the row order until you select it as the master index. As you update rows in a table, the affected indexes in the production index file are also updated. If you use any non-production .MDX files, they must be opened explicitly by entering statements in the Command window.

  The production index file has the same name as the table plus the .MDX extension.
- **Key expressions**. A key expression is a field name, or a combination of field names, functions, or operators, that determines how an index orders rows in a table. It must be a character, numeric, date, or float field, or an expression that evaluates to one of these types. The key expression can be up to 220 characters in length.
- **Primary key**. The dBASE 7 table format supports primary keys, enabling you to create primary distinct indexes. Any field can be a primary key and you need not create the primary key before creating a secondary maintained index.
- **Simple indexes**. A simple index uses a single field name for the key expression.
- **Complex indexes**. A complex (or composite) index uses a combination of one or more fields, or a dBASE expression.
- **Ascending and descending order**. Rows can be ordered in ascending order, lowest to highest (the default), or descending order, highest to lowest. For character fields, the order is ASCII or the order established by the language driver installed by the BDE.

Note

Keeping a large number of indexes affects performance, because dBASE PLUS must update each one as the table is revised. If you need to improve performance, consider removing rarely-used indexes from the production index file.

Planning indexes

When you design indexes for a table, consider how you will use and process data. Indexes affect and support features that an application provides: data entry, queries, and reports. Asking the right questions at the beginning can save you redesign efforts later.

- Using indexes in data entry

- Using indexes in queries
- Using indexes in reports
- Using indexes to link multiple tables

Using indexes in data entry

Because indexes affect the order in which rows appear, they let users find and update information quickly. To make data entry more efficient, consider these questions:

- What is the order in which users expect to see the data? For example, they might expect to see a list of companies in alphabetical order, a list of purchase orders by purchase order number, or a list of invoices in chronological order. Indexes should reflect the *expected* order of information in a table. If users expect the same information in different sequences, you can create multiple indexes—one for each sequence. For example, in the Orders table, you might want separate indexes for the order number, order date, and customer number.
- To find rows in a table, what kind of information might users know already? For example, to locate an invoice, users might already have the invoice number, approximate date of the invoice, or the company that submitted the invoice. To speed up the search process, you might want to create indexes for the most common ways a user looks for information.
- What kinds of calculations are users going to perform on data in the table? For example, users might want to calculate the average sale per state or the total sales per month. The word "per" is a clue to an index you might want to create—in the first example, indexing the state field and, in the second example, indexing the sales date field. An index can put similar rows in consecutive order so that users can quickly search for the first row in the series and stop processing after the last row in the series. For example, if users want to calculate the total payments to a vendor, consider creating an index for the vendor number or name.

Using indexes in queries

Indexes can increase the speed at which a query is processed. Indexes are also required for defining links among related tables. To make queries more efficient, consider the following issues:

- What kinds of questions are users going to ask? For example, will they want to know the number of items in stock for a particular product? If so, consider creating an index for the product name or identification number.
- What kind of information might a user know before attempting the query? For example, a user might know the name of the product, its identification number, or its type. Consider creating indexes for commonly known information.
- If the index is solely for occasional or ad hoc queries, consider generating an index at query time instead of maintaining an index separately on an ongoing basis. When the query is finished, you can delete the index to recover disk space.

Using indexes in reports

Indexes affect the order in which rows appear in a report. In addition, they can trigger subtotals and totals in a report (when key values change). To make reports easy to design, consider the following issues:

- What is the order in which users expect to see information in the report? For example, do users want to see a chronological list of invoices billed? An index can ensure that rows appear in the expected order.
- What kinds of calculations will the report make? For example, a report might show the total number of sales by salesperson, or the average sale by customer. The word "by" is a clue to an index you might want to create—in the first example, indexing on the salesperson field and, in the second example, indexing on the customer number. Using an index makes it easier to calculate running totals. If a report includes subtotals within totals, consider using a complex index.
- If the index is solely for occasional or ad hoc reports, consider generating an index at report time instead of maintaining an index on an ongoing basis. When the report is finished, you can delete the index to recover disk space.

Using indexes to link multiple tables

Indexes are required for linking related tables together in a multiple-table query. To link tables, consider the following issues:

- What are the relationships among the tables—one-to-one, one-to-many, many-to-many? For example, an Orders table and a LineItem table are in a one-to-many relationship. The Orders table is the parent table and the LineItem table is the child table.
- With related tables, which fields are common among them? To link tables together, you must have an index for the child table on a field that also appears in the parent table. For example, the Orders table and LineItem table both have an ORDER_NO field, and the LineItem table has an index on this field.
- Can you use codes instead of long character fields? For example, to link orders in the Orders table to customers in the Customer table, the application uses the customer number, a short character field that uniquely identifies each customer.

Creating a simple index

A simple index consists of a single field.

The key of a simple index is just the name of a field. For example, in the Customer table, if you index on the CUSTOMER_N field, the key is the field name, CUSTOMER_N.

You can create a simple index using either the Table designer or the Manage Indexes dialog box, as shown in the next two sections.

Using the Table designer to create a simple index

To create a simple index in the Table designer, choose an index order for the field you want to use—ascending or descending.

Using the Manage Indexes dialog box to create a simple index

To open the Manage Indexes dialog box, in Table design mode, choose Structure | Manage Indexes. The Manage Indexes dialog box appears.

To create a simple index,

1. Choose New. The Define Index dialog box appears.
2. Choose fields from the Available Fields list and add them to the Fields Of Index Key list at the right.
3. Choose Ascending or Descending order.
4. Choose Specify From Field List for a simple index.
5. Enter a name for the new index.

You can use letters, numbers, and underscores, but the first character must be a letter. The name you use must be unique within the index file. For a simple index, use the field name.

Check your vendor documentation for other limitations.

By default, dBASE PLUS indexes rows in ascending order. The exact sort order depends on the driver specified in BDE.

When you choose OK in the Manage Indexes dialog box, dBASE PLUS builds any indexes you created or changed and removes any indexes you deleted.

Note

You might have to wait while the indexes are created, particularly if the table has many rows or if key expressions are long and complex.

Selecting an index for a rowset

Depending on the table type, a rowset may be displayed in a form in different default orders. When you first open a dBASE table, it appears in natural order. When you first open a Paradox table, the natural order is the primary key order.

For dBASE tables, the production .MDX file opens automatically with the table, but the indexes it contains are not in effect until you select one.

To order rows that appear in a form in a specific way, select the index you want:

1. Open the form in the Form designer.
2. Select the active Query object.
3. In the Inspector, select the *rowset* property.
4. Click the *rowset* property tool button. The Inspector displays the rowset object's properties.
5. Set the *indexName* property to one of the available indexes.

Index tasks

In addition to creating and selecting indexes, there are several other index maintenance tasks.

Modifying indexes

You can modify an existing index to make it more useful or efficient. For example, if you create a simple index for a dBASE table in the Table designer, you might want to make it a complex index by adding fields or expressions. Or, you might learn after using the index for a while that a different key is more suitable.

To modify an index,

1. Open the table in the Table designer.
2. Choose Structure | Manage Indexes. The Manage Indexes dialog box appears.
3. Select the index you want to modify, and click the Modify button. The Define Index dialog box appears.
4. Make your changes, then choose OK.

Deleting indexes

You can delete an index you no longer need to save space and improve performance. Deleting an index does not delete any rows in the table—it deletes only the separate index that arranges rows in a particular order.

To delete a simple index, open the table in the Table designer, and choose None as the index type for the field.

To delete any other index,

1. Open the table in the Table designer.
2. Choose Structure | Manage Indexes. The Manage Indexes dialog box appears.
3. Select the index you want to delete, and click the Delete button.
4. Choose OK.
5. Save the table.

The index you deleted is removed from the production index file. If you delete the only index in the file, the.MDX file is deleted as well.

Indexing on a subset of rows for dBASE tables

In most cases, indexes include all rows in a table. For special circumstances, however, an index might contain only some of the rows in a table. Indexing on a subset of rows can make it easier to process information in that table. For example, you might want to work with budget information that applies to your sales department only. In this case, you could create an index that includes only those rows whose DEPT_ID is SALES.

To create an index that includes only the rows you want, first determine which rows you want to include, then state this in the form of a valid dBASE expression. For example, if you want to create an index of customers in your South sales region only, you could use a For *condition* expression such as SALES_REG = "SOUTH" to create the index. Thereafter, when you use this index, you see and process customers from the South region only.

Hiding duplicate values

Indexes can contain multiple rows with the same value in an indexed field. For example, a Line item table can contain multiple entries with the same ORDER_NO or STOCK_NO.

In certain cases, however, you might want to have a unique index, which finds only the first occurrence of a value in the indexed field and ignores subsequent rows with the same value. This kind of index is useful when subsequent rows repeat information in the first row.

For example, in a Lineitem table, if all products with the same STOCK_NO were sold at the same price, you could use a unique index to hide duplicate index values, so that only the first row with the price would appear.

If you check Include Unique Key Values Only in the Define Index dialog box, only the first row with a duplicate value in the indexed field is included in the index. Subsequent rows with duplicate values in that field are excluded.

Note

In dBASE and Paradox indexes, if there is a primary or distinct index, rows may not have duplicate values in the indexed field. Duplicate values cause an error when trying to save. In SQL indexes, uniqueness is required if the index is defined as a unique index.

Creating complex indexes for dBASE tables

Complex indexes on dBASE tables use a combination of one or more field names, plus valid dBASE expressions. Use a complex index when no single field uniquely identifies each row, or when you need the flexibility of an expression to define the index condition.

Indexes on .DB tables also can use multiple fields; such indexes are called *composite indexes*. However, unlike complex indexes in dBASE tables, you cannot use functions or operators in the .DB index expression.

Rules for dBASE complex indexes

For complex dBASE indexes, the complexity of the index expression varies according to the way the index is used. The following rules apply when defining complex indexes:

- An index value can be up to 100 characters long. The text of the key expression can be up to 220 characters long.
- The complex index must be a valid dBASE expression. Note that a single field name is a valid expression.
- The expression must evaluate to a character, date, numeric, or float value.
- It usually, but not always, contains at least one field name.
- For multiple character fields, *concatenate*, or combine, fields using the plus sign (+), as shown in the following examples:

  LAST_NAME + FIRST_NAME + M_INITIAL

  CUSTOMER + ORDER_NO
- You can concatenate fields of different data types by converting them to a single type. In the following example, the key expression concatenates the CUSTOMER_N field, which is a character field, and ORDER_DATE, which is a date field. The DTOS( ) function converts the date value to a character string in the format YYYYMMDD. This order—year first, then month, then day—ensures accurate indexing.

CUSTOMER_N + DTOS(ORDER_DATE)

- For converting number fields, use the STR( ) function. Include the width and number of decimal places of the numeric field(s), to ensure accuracy of the index. For example, suppose you are creating an index that includes a character field LNAME, and a numeric field called AMOUNT that is 10 places wide with 2 decimal places. Use the following syntax:

  LNAME+STR(AMOUNT,10,2)

Creating the dBASE complex index

To create a complex index for a dBASE table,

1. With the table open in the Table designer, choose Structure | Manage Indexes. The Manage Indexes dialog box appears.
2. Choose New in the Manage Indexes dialog box. The Define Index dialog box appears.
3. Select the combination of fields on which you want to index from the Available Fields list and move them to the Fields Of Index Key list. Or type a key expression, such as STATE+CITY, to create a complex index on the STATE and CITY fields. The key expression can use multiple field names, functions, and operators.
4. Click OK to exit the dialog box and save the index.

Key expressions

The following table shows several examples of key expressions and the fields used.

Table 13.2 Sample dBASE key expressions

Key expression	Fields used	Notes
CUSTOMER_N	CUSTOMER_N	
CUSTOMER_N + ORDER_NO	CUSTOMER_N, ORDER_NO	
CUSTOMER_N + DTOS(SALE_DATE)	CUSTOMER_N, SALE_DATE	DTOS converts date field to character for indexing.
UPPER(LAST_NAME)+UPPER(FIRST_NAME)	LAST_NAME, FIRST_NAME	UPPER changes character field to all caps.

The first example uses a single field as the key expression. Complex indexes, on the other hand, can use a combination of one or more fields, plus functions and operators.

- CUSTOMER_N + ORDER_NO is a complex key expression using multiple fields and the concatenation operator (+).
- CUSTOMER_N + DTOS(SALE_DATE) is a complex key expression consisting of multiple field names and a function.
- UPPER(LAST_NAME)+UPPER(FIRST_NAME) converts characters to all caps before concatenating them. The UPPER function prevents sorting problems when capitalized entries are mixed in with lowercase ones.

Primary and secondary indexes

dBASE PLUS lets you create primary and secondary indexes for any table type that supports them.

- A primary index is the main index in a table. For DBF tables, only level 7 tables support primary indexes; any expression index may be created as the primary index. For all other table types, the primary index consists of one or more consecutive fields, starting with the first field in the table.

- A secondary index is supplemental to the primary index in a table.

Some table types let you specify whether or not a secondary index is case-sensitive. Case sensitivity affects the sort order and the uniqueness of values. In dBASE PLUS, you can create case-sensitive indexes only, although dBASE PLUS maintains case-insensitive indexes when you edit tables that use them.

Each table should have one primary index, although it is not required. In a Paradox table, the primary index is stored in a file with a .PX extension.

Unique keys

Primary indexes require unique values—they do not permit duplicate key values. For example, if a dBASE table has a primary index on ORDER_NO, you cannot add two orders with the same order number—only one can exist in the table. In a composite index, individual field values can be duplicates, but the combined value of all key fields must be unique. (Secondary indexes do permit duplicate values.)

When you create the primary index, use a field that will contain a unique value for each row, such as a customer number field.

A table can have only one blank (empty) value in the keyed field, because subsequent blank values are considered duplicates. Therefore, key fields usually require entries.

Note

Some field types, such as memo, OLE, binary, and logical, are unavailable as key fields.

Secondary indexes, maintained and non-maintained

The dBASE Level 7 and Paradox table types permit two types of secondary indexes:

- *Maintained* secondary indexes are automatically maintained when data changes in the table. dBASE PLUS lets you create maintained secondary indexes, and it updates maintained indexes automatically when you edit a table.
- *Non-maintained* secondary indexes are not automatically updated when the table is open. dBASE PLUS does not let you create non-maintained secondary indexes, but it supports any existing non-maintained indexes.

The dBASE table format lets you create maintained secondary indexes regardless of whether the table has a primary index. You can create as many single-field (simple) indexes as there are fields in a table, and you can create up to 255 multiple-field (called *complex* or *composite*) indexes per table.

Creating primary indexes

You can create a primary index in the Table designer or the Manage Indexes dialog box. If the table type you are creating does not support primary indexes, these options are not available. dBASE Level 7 and Paradox table types both support primary indexes.

To create a primary index,

1. Open the table in the Table designer.
2. Choose Structure | Define Primary Key to display the Define Primary Key dialog box.
3. Choose the Primary Key fields from the Available Fields list. Click the arrow to add (or remove) fields from the Fields Of Primary Key list box.

Note

In the case of Paradox tables only, the first field in the table must be the primary key or part of a composite primary key. If the field you want to be the primary key is not currently the first one in the table, you have to move it up in the Table designer to be the first field. The dBASE format does not share this limitation.

Creating secondary indexes

You can create one or more secondary indexes in the Manage Indexes dialog box.

1. Choose Structure | Manage Indexes to display the Manage Indexes dialog box.
2. Click the New button. The Define Index dialog box appears.
3. Select fields from the Additional Fields select box and click the arrow to add each one to the fields Of Index Key box. The double-right arrow adds all the fields at once.
4. Choose Ascending or Descending order, assign a name to the indextag, and click OK.

Referential integrity

Referential integrity validates and updates the data in the linked key fields of a relational database. In a relational database, a field or group of fields in one table (the child table) refers to the key of another table (the parent table). Referential integrity rules ensure that only values that exist in the parent table's key are valid values for the specified fields of the child table.

You can establish referential integrity only between like fields that contain matching values. For example, you can establish referential integrity between two tables that both have a field that holds the customer number. The field names do not matter as long as the field types and sizes are identical.

dBASE PLUS lets you establish referential integrity for any file type that supports it, such as dBASE and Paradox table types. Some SQL-server tables also offer referential integrity. See your SQL-server database documentation to determine if your table type supports referential integrity.

The way referential integrity is used depends on the way you have set up indexing for the tables in a relational database. This section assumes you are familiar with the concepts of index creation and management.

Defining referential integrity

You can establish referential integrity between tables in the current database. If no database is specified, you can establish referential integrity between tables in the current directory.

To define a referential integrity relationship,

1. In the Navigator, Tables tab, use the Look In box to select a database alias or a directory containing tables (such as .DBF or .DB type) that support referential integrity.
2. Choose File | Database Administration. The Database Administration dialog box appears (To see SQL databases listed in this dialog box, you must first have given them a BDE alias.)
3. Specify a Table Type that supports referential integrity, such as dBASE or Paradox, then click Referential Integrity. The Referential Integrity Rules dialog box appears
4. Choose New. The New Referential Integrity Rule dialog box appears. All tables in the current database or directory appear in the Parent Table and Child Table drop-down lists
5. Choose a parent table from the Parent Table list. The table's key fields appear in the Primary Key Fields area of the dialog box.
6. Choose the child table from the Child Table list. Fields available for referential integrity appear in the Available Child Fields list.
7. Specify whether the tables are in a one-to-one or one-to-many relationship in the Relationship panel. The relationship you choose changes the available child fields.
   - One-to-one relationships can be defined between the primary key field in the parent and the primary key field in the child, or any field in the child that has a unique index.
   - One-to-many relationships can be defined between an indexed field that is not the primary key in the child and the primary key field in the parent.
8. Choose the child table's field in the Available Child Fields list and click the Add Field arrow. The field name appears in the Related Child Fields area of the References panel.

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

9. Select the update and delete behavior you want (see below).
10. Optionally change the rule name dBASE PLUS provides in the topmost box.
11. Choose OK to save the referential integrity relationship.

Note

If you attempt to define referential integrity on a table that already contains data, some existing values may not match a value in the parent's key field. When this happens, the operation fails and you receive an error message.

Update and delete behavior

You can specify the following rules for updating and deleting data in a parent table that has dependent rows in a child table:

- **Restrict:** You cannot change or delete a value in the parent's key if there are rows that match the value in the child table.

  For example, if the value 1356 exists in the Customer No field of Orders, you cannot change that value in the Customer No field of Customer. (You can change it in Customer only if you first delete or change all rows in Orders that contain it). If, however, the value doesn't exist in any rows of the child table, you can change the parent table.

- **Cascade:** Any change you make to the value in the key of the parent table is automatically made in the child table. If you delete a value in the key of the parent table, dependent rows in the child table are also deleted.

The availability of cascading updates and deletes varies according to the table type:

- **dBASE Level 7**: Cascading updates or deletes permitted
- **Paradox**: Cascading updates only
- **Oracle**: Cascading deletes only
- **Sybase**: No cascading updates or deletes permitted
- **InterBase**: No cascading updates or deletes permitted
- **Microsoft SQL Server**: No cascading updates or deletes permitted

Changing or deleting referential integrity

You can choose any referential integrity name from the list of named referential integrity relationships in the Referential Integrity Rules dialog box to either modify or delete it.

- Choose Edit to open the Edit Referential Integrity Rule dialog box with the selected referential integrity relationship filled in. You must be able to obtain exclusive access to all tables involved in the referential integrity when you modify it.
- Choose Drop to delete the selected referential integrity relationship.

Chapter 14 Editing table data

Chapter

14

Editing table data

To browse, change, or add to data in a table, open the table in Run mode. You can use any of three different layouts: grid, form, and columnar.

Though the various data sources supported by dBASE PLUS have different capabilities, limitations and structures, the same basic procedures for running tables from within dBASE PLUS apply to all.

This section describes how to use dBASE PLUS's built-in data-editing capabilities to

- Scan information
- Find or replace information
- Perform data entry (add information)
- Delete and undelete information
- Save or abandon changes
- Operate on a subset of information
- View and edit special field types

A few words of caution

Like any file, tables and the information they contain can be corrupted or destroyed if used improperly.

If, for example, you design an application that prevents entry of a number greater than 10 in a field called NUMITEMS, and the existing data is contained in an older .DBF table, someone could circumvent your data constraint of "not greater than 10" by opening the table in Run mode, adding a new row with a value of 11 in the NUMITEMS field, and saving the table.

Most table types, including the new .DBF7 format, allow the table developer to enforce rules at the table level in order to prevent such problems from occurring. But even with such safeguards, developers should caution users who have access to tables that database integrity can be compromised by editing table data directly in dBASE PLUS or in any other application—including spreadsheets and word processors.

Developers can also take other simple precautions, such as placing data in folders at levels casual users may not venture into, or naming data directories with numbers instead of with tempting titles like "databases."

It also helps to understand exactly when and how your databases can be opened and modified from within dBASE PLUS.

Running a table

To view or edit a table, open a table in Run mode using any of the following methods.

- Menu: Choose File | Open (Alt+FO). The Open File dialog box appears. If it's not already selected, choose the Tables (*.dbf, *.db) item from the Files Of Type list and locate a table on your local drives, or select an alias from the Database list. Then choose the View Table Rows option at the bottom of the dialog box. Select your table, then click Open.
- Project Explorer: Select a table, right-click, and choose "Run" from the context menu.
- Navigator: Choose the Tables tab, then use the Look In drop-down list to choose a local folder or alias (or use the Browse button to choose an unlisted folder). Then double-click a table icon. You can also open a selected table by clicking the Run button on the toolbar or by pressing F2.
- Command window: type

  use <tablename>

  where <tablename> is a local table file name or aliased :database:table reference, for example,:MSSQL1:mytable. You may include the full path to the file name. Then type

  edit

The table appears in a grid of rows and columns. Each column is a field. You can browse or edit all the data in the table.

Protected tables

When you open a protected table, complete the Group, User, and Password fields in the Login dialog box, and choose OK. The database administrator assigns groups, users, and passwords for table protection.

Also note that some tables may have read-only protection or other access and editing restrictions.

Search the Help index for "security" for more information about protected tables, access rights, and encryption.

Table tools and views

When you run a table or query, additional items appear on the View menu, and a Table menu becomes available.

Figure 0.1 Table-editing toolbar

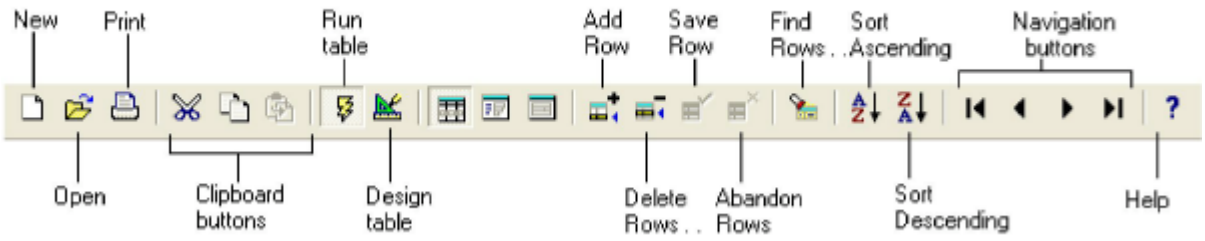


Table and query views

The default view when running tables or queries is a grid view. Other options are a columnar view, which displays a single row on each page with fields arranged vertically, and a form view.

You can choose your preferred view any time in two ways:

- Choose the desired view from the View menu.
- Click the appropriate button on the toolbar: Grid Layout, Columnar Layout, or Form Layout.

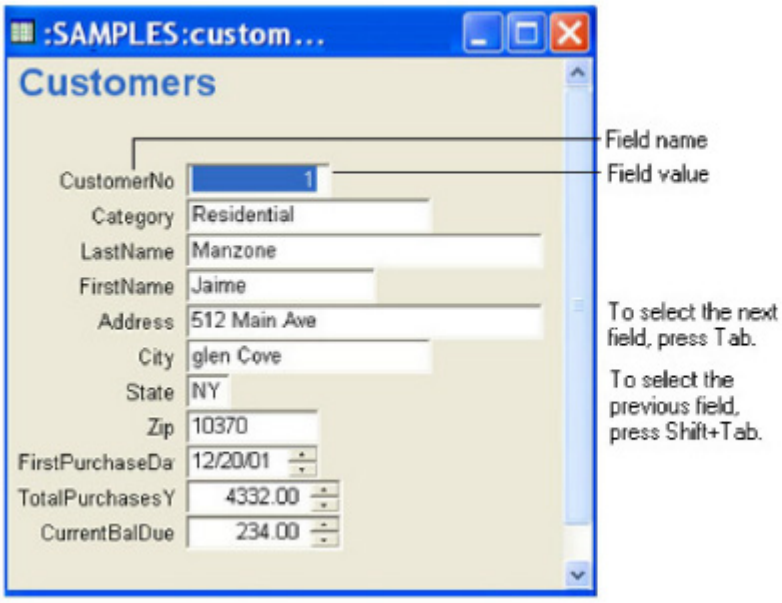
dBASE PLUS remembers your last view choice and next time you open a table, it will be in the view you last chose.

Adjusting the view

In columnar view, fields remain at their default size and cannot be widened. You can, however, enlarge the view window vertically to see more fields at once.

- In grid view, you have other options:
- You can resize columns and rows by pointing to a column or row border, then dragging when the pointer changes to a double-headed arrow.
- You can move columns by dragging field titles to new positions.

Figure 0.1 Columnar view



Viewing only selected table data

To open a table with only specified rows available for viewing or editing, create and run a query—an SQL statement that requests specified information from one or more tables.

To view or edit data selected by a query, open the query in Run mode using any of the following methods.

- Menu: Choose File | Open (Alt+FO). The Open File dialog box appears. If it's not already selected, choose the SQL (*.sql) item from the Files Of Type list and locate a query on your local drives. Then choose the Run SQL option at the bottom of the dialog box. Select your query, then click Open.
- Project Explorer: Select the query you want to run, right-click it, and choose Run from the context menu.
- Navigator: Choose the SQL tab, then use the Look In drop-down list to choose a local folder (or use the Browse button to choose an unlisted folder). Then double-click a query icon. You can also open a selected query by clicking the Run button on the toolbar or by pressing F2.

The query results appear in a grid of rows and columns. What you edit here is reflected in the table or tables that contain the data.

For more information on creating, modifying and running queries, search for "SQL" or "queries" in the Help index.

Table navigation

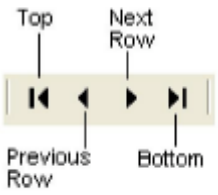
dBASE PLUS uses a row pointer to identify the current row. Use the following methods to move the row pointer in a table:

Table 14.1 Navigating rows using the menu, mouse or keyboard

Move to	Menu	Mouse	Keyboard
Next row	Table | Next Row	Click next row	PgDn (columnar)
Previous row	Table | Previous Row	Click previous row	PgUp (columnar)
First row	Table | First Row	Scroll to top of table, if necessary, and click first row	Ctrl+Home
Last row	Table | Last Row	Scroll to bottom of table, if necessary, and click last row	Ctrl+End
Specific row	Table | Find Rows	Click row	Ctrl+F
Previous page	Table | Previous Page	Click in the scroll bar	PgUp (grid)
Next page	Table | Next Page	Click in the scroll bar	PgDn (grid)

In addition, you can use the navigation toolbar buttons.

Figure 0.1 Navigating rows using the toolbar



Note

Your Desktop Properties settings might cause the exclusion of certain rows in a table. For example, if Deleted is selected on the Table page in the Desktop Properties dialog box, the row pointer skips deleted rows. Similarly, if you've specified a scope in the Table Rows Properties dialog box, the row pointer ignores rows outside the scope. If you've specified a filter for the table, rows not meeting the filter condition are ignored. For details on these and other Table Property settings, see Help (search for "table properties").

Data entry considerations

When you're faced with day-to-day data entry tasks, consider the following:

Should I run a table or use a form? Running a table offers quick direct access to tables from within dBASE PLUS. This is handy for occasional editing, data entry, and maintenance. However, forms offer more control over the data-entry process, including the ability to edit multiple linked tables in the same window and to programmatically enforce entry validation and data integrity. If a data-entry form doesn't already exist, you can use the Form wizard to quickly create one. For ongoing data entry and maintenance, consider designing a form.

Editing all rows or selecting only the information you need. You sometimes want to work with only a subset of rows in a table, especially if the table has a large number of rows. For example, you might want to change orders for the current month only. Consider the following approaches:

- Use queries to select the rows you need to change and ignore the rows that don't apply to the task at hand. One advantage to using queries is that they allow you to store the conditions you specify and use them with multiple tables.
- Use a conditional or unique index that includes only the rows you want.

Working with parent and child tables. Deleting rows or changing the values in linked fields or key fields can cause dBASE PLUS to lose track of data. For example, if you delete an order in the Orders table but not in the associated rows in the LineItem table, you end up with orphaned rows that could skew calculations. Similarly, you might inadvertently change the order number in the Orders table but not in the LineItem table, which also results in orphaned rows.

If the table you are editing is part of a parent-child relation, consider using a query to link the parent and child table, rather than editing the single table. The query helps you see and preserve connections between related rows in the tables.

Ordering rows. During data entry, you can use the natural order of the table or you can use an index. When searching for rows to update, using an index could be the most efficient means, particularly in a table with many rows.

Selecting a view for entering data. When you run a table, three views are available: grid (default), form and columnar. Choose the one that best suits your data entry task.

Repeated values. If you are entering the same value repeatedly, consider using the Replace option to update a number of rows with the same value quickly.

Note

The Data Entry page of the Desktop Properties dialog box offers a number of data entry configuration options, including Bell, Confirm, Delimiters, and Type-ahead. For details on these options, click the Help button on the Data Entry page.

Finding and replacing data

dBASE PLUS provides tools that let you search for information in a table and update rows with new information.

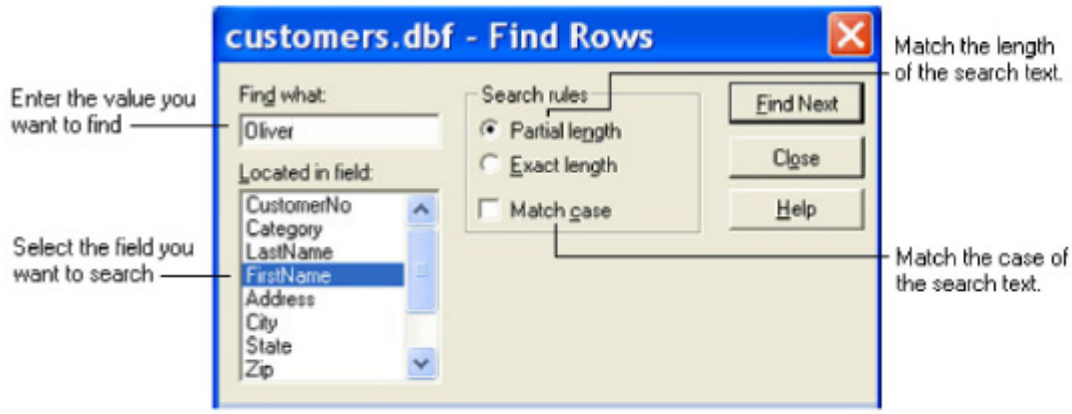
Searching tables

In addition to scrolling through rows, you can quickly find the row you want by searching for a value in a field you select. For example, you could quickly find a specific customer order by selecting the ORDER_NO field and typing the number of the order you want to find. You can search character, numeric, float, date, and memo fields.

To begin a search, click the Find Rows button, or choose Table | Find Rows.

The Find Rows dialog box provides options you can use to focus and speed up your search. The options you use depend on the search value you specify, the way information is organized in the table, how specific the search needs to be, and how much of the table you want to search.

Figure 0.1 Find Rows dialog box



- **Find What** In your search text, you can specify any printable character, including spaces. The search string can be as long as the width of the search field. In general, the longer the search string, the greater the precision required. If you can't find a match with the current search string, shorten it to increase your chances of finding a match.
- **Located in Field** You can search for text in any field, whether or not it has been indexed. Searching is fastest when you search on an indexed field. Before you start your search, select the index you want to use as the master index.
- You can also search non-indexed fields, such as memo fields. Doing so might be slower than an indexed search, particularly in tables with many rows.
- **Partial Length** There is no requirement that the length of the search string be identical to the field value. This rule is checked by default.
- **Exact Length** To be a match, the search string must appear in the field just as you type it.
- **Match Case** Match Case requires that the field value match the search string exactly, including uppercase and lowercase letters.

Once you have selected the options you want, click Find Next. If a match is found, the row pointer moves to the matching row and the row appears highlighted. If no match is found, a message appears.

If you don't find the match on the first try, shorten the search string or adjust other search options as needed and try again.

Replacing data in rows

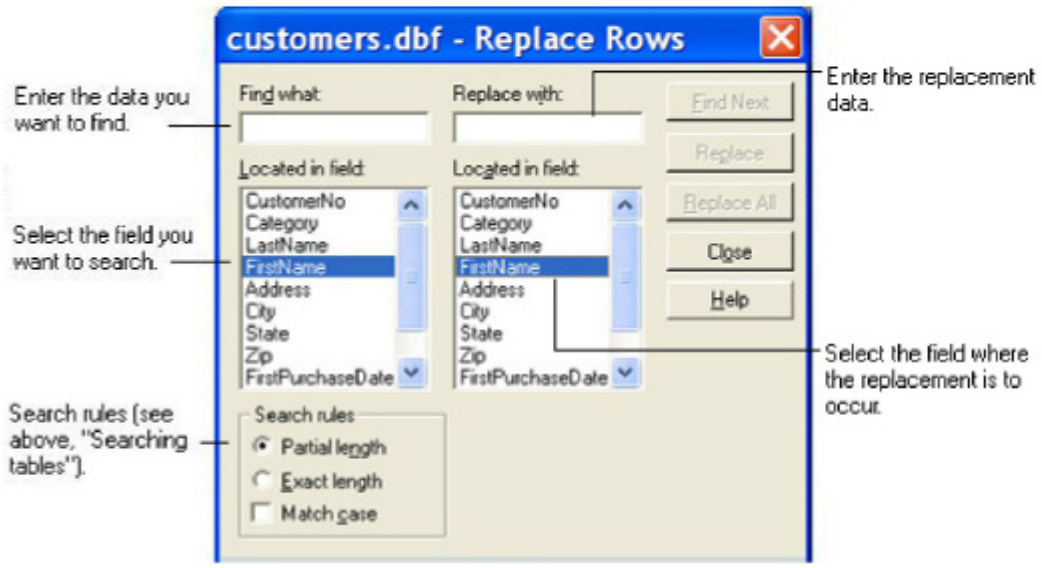
You can find text in a table and replace it with different text. For example, if you change the name of a product, you can search a table and replace the old name with the new one. The replacement can occur in a different field than the find string is in.

For example, suppose you assign a salesperson to a different sales territory and want to update the SALESPERSON field for all customers in that territory. Rather than update each customer row individually, you could simply select all the customer rows in that territory, then replace the SALESPERSON field in those rows only.

Important

Updating indexed fields in the master index can yield unpredictable results because changing the key value changes the row position—as well as the row pointer—in the index. Use a different master index instead. In addition, changing key values in related tables can result in orphaned rows in the child table. Therefore, carefully consider the implications of updating rows, and make a backup copy of your tables before proceeding.

Figure 0.1 Replace Rows dialog box



To replace rows,

1. Select the table, then choose Table | Replace Rows. The Replace Rows dialog box appears.
2. Complete the dialog box.
   - The replacement value you specify must match the data type of the selected replace field. Make sure that the value fits in the field.
   - In character fields, if the text is too long for the field, it is truncated.
   - In number fields, if the value exceeds the field size, the fields fill with asterisks.
   - In memo fields, the existing memo text is overwritten with the replacement text. The replacement text must be in character format.
3. Once you've specified the replacement text, do one of the following:
   - Choose Find to find the next occurrence of the search text. Then choose Replace to replace it, or choose Find to leave it alone and go to the next occurrence.
   - Choose Replace All to replace all occurrences of the search text.

Adding rows to a table

When you add rows to a table, they are appended to the end of the table. An empty row is added at the end of the table where you can enter data. If a table contains 100 rows before you append, the new row becomes row number 101 and the current row for editing.

To append a row, do one of the following:

- Choose Table | Add Row.
- Click the Add row button on the tool bar.
- In grid view, go to the last row, and then press the Down arrow.

An empty row appears. Now you can enter data.

Note

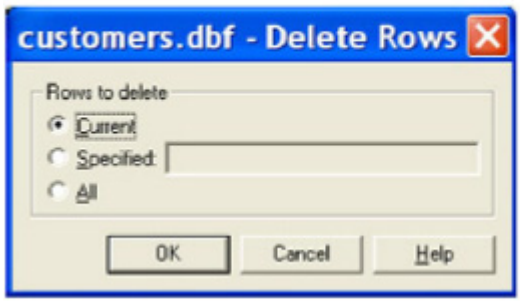
If you're adding rows to a table with an active index, each row appears at the end of the table. When you finish entering data in the row, dBASE PLUS updates the index and moves the row to its correct position as indexed. The last row you added remains the current row for editing.

Deleting rows

To delete a row in grid view, select the row by clicking the button at the left of the row, then press Ctrl+U. In columnar view, navigate to the row and press Ctrl+U.

1. You can also delete rows through the Delete Rows dialog box:
2. Find the row in the columnar view or select the row in the grid view.Choose Table | Delete Rows, or click the Delete button on the toolbar. The Delete Rows dialog box appears
3. Click OK to delete the currently selected row, or specify a particular row number, or choose all to delete all rows. Click OK.

Figure 0.1  ows dialog box



Saving or abandoning changes

dBASE PLUS saves changes to a row automatically whenever you do one of the following:

- Move the row pointer to another row.
- Toggle the table view between grid and columnar view.

Note

If Autosave is selected in the Table page of the Desktop Properties dialog box, dBASE PLUS writes changes to disk automatically. Otherwise, it accumulates changes and saves them to disk periodically.

- To save a row manually, do one of the following,
- Close the Table window, or
- Click the Save Row button

To abandon changes to a row, click the Abandon Row button.

Performing operations on a subset of rows

Sometimes you need to work with only a subset of the rows in a table. You can save time by selecting only the rows you want to work with, and avoid processing rows that don't apply.

This group of topics explains how to

- Select sets of rows to process
- Count rows that meet a given set of criteria

- Perform calculations on multiple rows
- All of these features are found in the Table menu.

Many of the operations described in this section—including aggregate operations for calculating data, finding rows, filtering conditions, and linking parent and child tables—can also be performed by creating queries. For information, search the Help index for "SQL" or "queries."

Selecting rows by setting criteria

To select a subset of rows from your table, you have to identify the criteria by which rows qualify for selection.

For example, if you want to calculate the average sales volume of customers in Texas, your criteria might be that the STATE_PROV field in the Customer table contain the value "TX". Rows that fail to meet this criteria are ignored.

Another row selection technique is to specify a condition by writing an expression in the Command window that defines which rows qualify for processing.

Setting For conditions

Use the For condition to select rows that appear throughout a table rather than in a contiguous group. For conditions, check all rows in the table when determining which rows qualify for processing. Processing starts at the top of the table and goes to the bottom (unless you limit it using one of the options discussed in the previous section).

dBASE PLUS compares each row with the condition you specify to determine whether to process a row. For example, to count the number of customer orders that exceed $10,000, you might specify the following For condition: TOT_INV > 10000.

Setting While conditions

Use the While condition to select a series of rows that appear consecutively in a table. While conditions check only the current row and subsequent rows when determining which rows qualify for processing. Processing starts at the current row rather than at the top of the table. Therefore, the row pointer must be at the first qualified row before processing; otherwise, no rows can be selected.

This method works best when you use an index whose key matches fields in the condition. That way, you can quickly find the first row in the series, then process rows sequentially. Processing ends when the key no longer matches the condition.

For example, to do a calculation only on rows of customers in Texas, it would speed up processing to apply an index on STATE_PROV, which would group all the TX rows together. Search for the first TX row, and then enter STATE_PROV="TX" in the While field to process from the current row through the last TX row.

You can also create or modify an index to include a subset of rows. In this case enter a For condition that specifies STATE_PROV="TX". When that index is active, only TX rows are selected.

Counting rows

You can count rows to determine how many rows meet a given set of criteria. For example, you might want to know how many customers fall within a certain zip code range, or how many orders were taken on Tuesday.

To count rows:

1 Choose Table | Count Rows. The Count Rows dialog box appears.

Figure 0.1 Count Rows dialog box

2 Specify the rows to include in the count, then choose OK.

dBASE PLUS counts the number of rows that meet the criteria, then displays that number in a message box.

Performing calculations on a selection of rows

You can perform calculations on number fields to obtain useful information from a selection of rows. For example, you might calculate the total sales for a given month, the largest sale, the smallest sale, or the average sale amount.

You can perform the calculations shown here

Table 14.2 Types of calculations

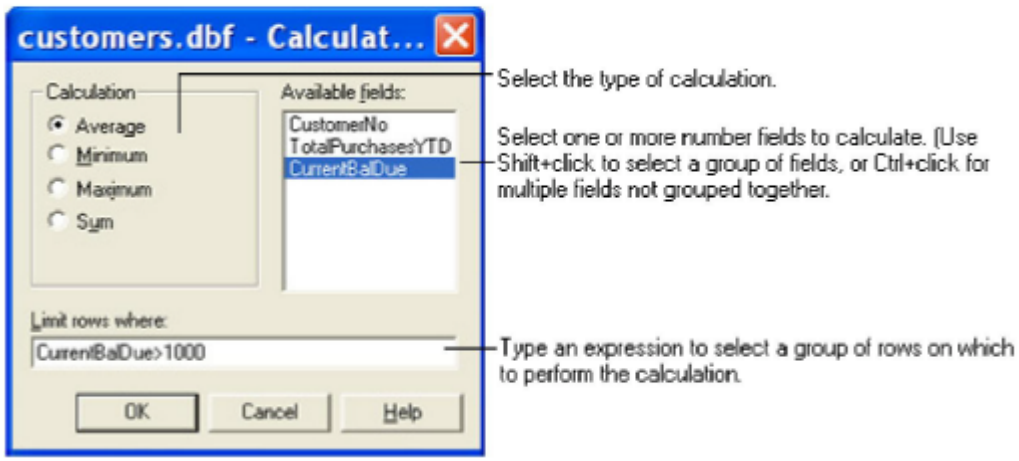
Calculation type	Result
Average	Average field value in selected rows
Minimum	Minimum field value in selected rows
Maximum	Maximum field value in selected rows
Sum	Sum total of field values in selected rows

Most calculations work on numeric and float fields only, but Maximum and Minimum can also be used with date and character fields.

To calculate values:

1 Select the table, then choose Table | Calculate Aggregates. The Calculate Aggregates dialog box appears.

Figure 0.1 Calculate Aggregates dialog box



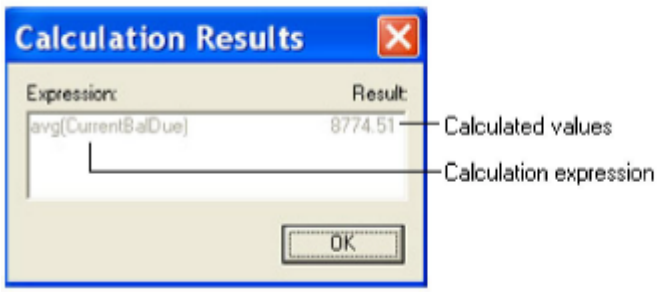
2 Choose the type of calculation you want to perform, then select one or more of the listed fields. (Use Shift+click to select several contiguous fields, or Ctrl+click to select several noncontiguous fields.)

   Optionally, you can type an expression in the Where box to select a group of rows on which to perform the calculation

3 Click OK.

dBASE PLUS performs the calculation and displays the results in the Calculation Results message box.

Figure 0.1 Calculation Results dialog box



Viewing and editing special field types

Supported field types and field-editing rules can differ greatly from database to database. In many cases, entering and editing data is intuitive enough: select a field and type in characters or numbers, depending on the field type. If you enter a value that doesn't match the basic field type rules (like entering a character in a numeric field), you'll receive an error message.

Most databases also offer a number of non-character field types, however, and viewing and modifying data in these types is a bit different.

This group of topics examines binary (image or sound), OLE, and memo field types, all of which are available in both the older .DBF and the new .DBF 7 table formats. Other table formats may offer other ways to edit similar field types.

Viewing the contents of special field types

Memo, image, sound, and OLE fields are represented in a table by icons. You can view the contents of these types of fields in three ways:

- Select the field and press F9.
- Double-click the field.
- Select the field and choose View | Field Contents.

To select a field with the keyboard, use the following keys.

Table 14.3 Field selection keyboard shortcuts

Go to	All views
Next field	Tab
Previous field	Shift+Tab
Beginning of field	Home
End of field	End

Memo fields

Memo fields open in a text editor. When the text editor is open, the Format toolbar becomes available, and you can format text in the memo field.

Binary fields

Your tables can contain any supported sound and image data, and the data can be stored, viewed (or played, in the case of sound files), added, or replaced any time you run a table.

.DBF and .DBF 7 tables support most popular image formats (for a complete list, see the Image page in the Navigator). The supported sound format is .WAV.

Importing an image or sound into a binary field

To add an image or sound to a binary field,

1. Double-click a binary field. The Specify Binary Field Subtype dialog box appears.
2. Choose the binary type (Image or Sound). If you choose Sound, the built-in Sound Player appears. If you choose Image, the built-in Image Viewer appears.
3. Right-click on the viewer or player. A file import dialog box opens. Select a file of the appropriate type, then click OK.
4. Save the row.

If the field already contains a binary image or sound, you can export the image or sound to disk by calling the player or viewer as instructed above, then choosing Export from the right-click menu.

OLE fields

Object linking and embedding (OLE) lets you use objects from other Windows applications in your tables.

You can either link objects to or embed objects into OLE fields. Linking inserts a reference to the file from which the object originated, which means that in order to keep the object updated, both the source file and source application must remain available. If the linked object is updated, your OLE field is updated as well.

Embedding places an entire object into the OLE field. Embedding is a more portable solution, but still requires that the application that created the source be available. It can also cause significant enlargement of your table file sizes, which grow by the size of each object you embed plus some OLE reference code for each. And unlike links, embedded objects are not updated when the source object changes. Instead, they become separately editable (in the source application) objects of their own.

An OLE object can be a graphic image, a sound, a document created by a word processor, or any other object or document that can be created by an OLE-compliant server application. For example, Microsoft Word is an OLE server, and any document created in Word can be linked or embedded into an OLE field.

In any OLE exchange, dBASE PLUS then becomes the client application.

Whether you choose to link or embed an object into your OLE field, you can launch the server application and load the object for editing by simply double-clicking the OLE field in your running table.

Adding an OLE object to an OLE field

To add an object to an OLE field

1. Start the server application and open a file or create an object.
2. If you want only a portion of the object data, select it and copy it to the Clipboard using the application's Copy command or Ctrl+C.

3. Start or switch to dBASE PLUS, and run the table containing an OLE field.
4. Click the Add Row button or choose Table | Add to open a new row. If the OLE field is represented by an icon, double-click the icon to open the OLE viewer.
5. Do one of the following:

- To link the object, right-click the OLE viewer and choose Paste Link from the popup menu.
- To embed the object, right-click the OLE viewer and choose Paste (Ctrl+V) from the popup menu.

The linked or embedded object appears in your OLE viewer. You may need the scroll bars to scan the entire object.

A linked object is automatically updated by default, so if the source object is edited, the OLE field will reflect the changes. You can modify link attributes, however—as well as view information on the link, open the source file, change the link (useful if the source file is moved) or even break the link—by right-clicking the OLE viewer and choosing Links from the popup menu.

To edit an embedded object, double-click the OLE viewer containing the object. The server application opens with the object loaded for editing. When you're finished with your edits, update the OLE field by choosing Update from the server application's Edit menu.

You can also link or embed OLE objects by right-clicking an OLE viewer window and choosing Insert Object from the popup menu. The Insert Object dialog box lets you choose from among the OLE object types registered on your system. You can then either create a new object in the server application (for embedding) or create an object from an existing file (for embedding or linking).

Removing an OLE object from an OLE field

To remove an OLE object from a table,

1. Locate the OLE field that contains the data you want to remove.
2. Double-click to open the OLE viewer (if it's not already open).
3. Select the viewer window and choose Edit | Delete.

Chapter 15 Security

Chapter

15

Setting up security

dBASE PLUS provides built-in levels of security against unauthorized access to encrypted databases and tables. This table-level security depends on data encryption.

Sensitive tables should always be encrypted by using the database vendor's administration software. dBASE PLUS's password dialog is presented whenever a user tries to access a form linked to an encrypted table or database. The user's response to the password dialog is passed to the encrypted table or database for verification before dBASE PLUS will display the form. See your database vendor's documentation about security administration for SQL, ODBC, or non-Standard systems.

The DBF and DB tables you create within dBASE PLUS have built-in encryption. dBASE PLUS provides direct database administration security access to set passwords for BDE-Standard DB and DBF tables, as well as the extensive user-access and privilege-level security features of DBF tables.

IMPORTANT NOTE: when using ADO to connect to other database tables the built in security options in dBASE PLUS IDE will not work. You will need to use the security setting on your DBMS server instead.

Setting up security strategies

dBASE PLUS offers two general strategies to handle access to encrypted tables of any type: individual login and preset access.

- Individual login via automatic password dialogs
  In this approach, each user is required to login every time he or she tries to access a form linked to an encrypted table or database. dBASE PLUS automatically displays a password dialog for the appropriate table type, requiring the user to enter a password or other information required by the table. Users might get different access levels, depending on their user name and password, and depending on the security features supported by the table type. The user must submit the correct information (which is passed to the encrypted database system for verification) before the user can access the dBASE PLUS form.
- Preset access via Session and Database objects
  Preset access involves hard-coding passwords or user names in dBASE PLUS forms and reports. Preset access provides an automatic, pre-determined level of access without login procedures for certain groups of users. It can

be used in conjunction with individual login to provide easy read-only access for the public and login-protected access for authorized company personnel.

  - Preset access for Standard table types
    Sessions objects provide unique connections between a user and a DBF or DB table. You can add methods to these objects to restrict access to certain features of a Standard table, or to make the table read-only for certain login-levels.
  - Preset access for SQL and other table types
    Database objects link dBASE PLUS forms to SQL databases or table sets. You can set the Database object's *loginString* property for particular user names or passwords to limit users of a specific login-level to read but not write the data in an SQL database.

- Table-level security for DBF tables
  dBASE PLUS supports direct access to the extended security features of DBF tables, including administrator security, up to 8 user access levels, and three-level privilege security for DBF tables and individual fields. If you intend to create tables within dBASE PLUS, DBF tables offer the most extensive and versatile security features.
- Table-level security for DB tables
  dBASE PLUS provides direct access to master password security for each DB table. However, you must use Borland's Paradox or Database Desktop to set auxiliary passwords.

Individual login via automatic password dialogs

dBASE PLUS's password dialogs are activated only by encrypted tables. Password protection alone is inadequate to protect a sensitive table unless the table is encrypted, because an intruder, having gained access to the machine or server on which the application is running, could use another application to read the data from the hard disk.

Once an authorized user gains access to your application by providing the correct password, the user might be offered a restricted choice of a variety of tables, with different access privileges, depending on the login level used to access the application. dBASE PLUS supports the full range of table- and row-level security features for DBF tables, so you can create up to 8 user access levels and 3 privilege levels, precisely controlling access by different classes of users to specific tables and even to specific rowsets in those tables.

To link any encrypted table to a form (and thereby enable automatic password protection), you need only create a Query object for that table on your form. To do this, simply drag the table icon of the encrypted table from the Navigator's Tables tab to your form surface in Form designer. (At this time, while in Design mode, you will have to supply access information to the encrypted table.) This is all you need to do to ensure that dBASE PLUS will activate the automatic password dialog. See Form Designer for guidance on adding Query objects to forms.

After your form is run and the user activates some event (a button click, for example) to access a restricted table, the Borland Database Engine (BDE) attempts to open that table. Because the table is encrypted, dBASE PLUS automatically displays the appropriate password dialog with fields for the input required by that table type. The type of security available varies according to table type.

The original dBASE PLUS form is temporarily displaced and the user is presented with a password dialog. To gain access to your application (and its underlying encrypted table), the user must provide the particular security information required by that table or database.

Preset access via Database and Session objects

Setting preset access levels is another approach to restricting access to your data.

To set levels, you have to hard-code passwords or user names in dBASE PLUS forms and reports, thus restricting access only to individuals within specified groups.

Preset access can be useful in combination with the individual login approach to provide easy read-only access for general users, and login-protected access for authorized personnel. For example, you might have employee information managed through an application that allows updating by Human Resources personnel, but permits read-only access to other employees.

To implement this strategy, you would need a full-access (read/write) password that the HR staffers would have to enter manually every time they start the application. You would then code into the form a read-only password that would admit everyone else at a limited level.

How you encode preset access levels depends on the table type, as described in the next two topics.

Preset access for Standard table types

For DBF and DB tables, security is session-based. The Session object has a *login*(_) method for DBF table security and a *addPassword*(_) method for DB table security. The appropriate method (or both if you're using both types of encrypted tables) must be called with the correct user name and password before attempting to activate a query that accesses the table.

Where this must occur depends on whether all users are sharing the same session. If everyone accessing dBASE PLUS gets exactly the same access for every application by using the same user name and password, then they can share the default session. You would need only call the session's methods once through an administrative program or form.

All the forms would thus require a Query object to access the DBF or DB tables, but no Database object or Session object, because everyone uses the default database in the default session.

On the other hand, if any two applications use different user names or passwords, then every form must have its own Session object, so that each form runs in its own session and the security is localized.

No Database object is needed because the form uses the default database of its own session. Then users must log into each session before the query is activated.

Use the Query object's *canOpen* event to call the session's security methods.

Preset access for SQL and other table types

Table and database types other than DBF or DB tables accessed via the directory require modification of the Database object's loginString property. This applies to all non-Standard applications, including SQL servers such as Borland InterBase, Oracle, Sybase, Informix, IBM DB2, and MS SQL Server; and ODBC connections such as Access and Btrieve. It also applies to remote DBF and DB tables accessed through a Borland Database Engine (BDE) alias.

A BDE alias always identifies a database. Therefore all non-Standard table security is through the Database object that provides access to that database. In some cases, logins are required to access tables in a database.

The Database object's *loginString* property is a character string that contains the name and password in this form:

```
name/password
```

You can set this property in the Inspector in the Form designer. By setting the name and password in the form's Database object, all users attempting to open that form will get whatever level of access that name and password provides.

Although possible, it's more trouble than it's worth to share a Database object among multiple forms. Each form should have its own Database object, with whatever the appropriate *loginString* is for that particular form.

Table-level security for DBF tables

The security features of DBF tables are extensive. If you intend to create private tables within dBASE PLUS for which you want to set elaborate or varied access levels, the DBF table type is your best choice.

Table-level security relies on data encryption. Data encryption scrambles data so that it can't be read until it is unscrambled. An encrypted file contains data that has been translated from source data to another form that makes the content unreadable. If your database system is protected, dBASE PLUS automatically encrypts and decrypts

tables and their associated index and memo files when a user supplies the required passwords or other login information.

In addition, DBF tables allow you to define which fields within tables users can access, and the level of access, read, read/write, or full.

The first parts of this section describe how to plan your security scheme for DBF tables. Topics include

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

At the end of this section are procedures for setting up your DBF security scheme:

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

About groups and user access

You can control access to individual DBF tables (and to fields within those tables) by carefully defining groups of users according to

- Which tables each group can access
- Which privilege levels (read, update, extend, delete) each group has at the table-level
- Which fields within tables each group can access
- Which privilege levels (none, read-only, full) each group has at the field-level

Table access

First, you'll need to define user groups and determine which group has access to which table. Try to organize users and tables into groups that reflect application use (for example, by department or sales area).

- A table can be assigned to only one group. If the user group and table group don't match, the user can't access the table.
- Typically, each group is associated with a set of tables. By associating each application with its own group, you can use the group to control data access.
- A user can belong to more than one group. However, each group that a user belongs to must be logged-in separately.
- If a user needs to access tables from two different groups in the same session, the user must log out of one group, then log in to the second. A user may have separate logins into different groups in separate sessions to access files in different groups.

User profiles and user access levels

You'll need to develop a user profile for each user in each group. As part of each profile, you'll assign to the user an *access level.* Each user's access level is matched with the table's privilege scheme (see the next section) to determine what access the user has to the table and, within each table, to each field. For example, if you establish a

read privilege of 5 for a table, users with a level from 1 to 5 can read that table. Users with a level of 6 or higher can't read the table.

By establishing access levels within a group, you can give different users different kinds of access to the table and to fields within the table.

- Access levels can range from 1 to 8 (the default is 1). Low numbers give the user greater access; high numbers limit the user's access. The access value is a relative one—it has no intrinsic meaning.
- The less restrictive levels (1, 2, 3) are typically assigned to the fewest people. To limit access to data, the more privileges a level has, the fewer users you should assign to that level.
- You can assign any number of users to each access level.
- If you don't need to vary the access level of the users within a group, there is no need to change each user's default level.

About privilege schemes

Once you've established each user's access level, you set up a *privilege scheme* for each table. A DBF table's privilege scheme controls three things:

- Which group can access the table. (The user's group name is matched with the table's group name to allow table access.)
- Which user access levels can read, update, extend and/or delete the table (table privileges).
- Which user access levels can modify and/or view each field within the table (field privileges).

After a user logs in, dBASE PLUS determines what access the user has to that DBF table and its fields by matching the user's access level with the rights you specified in the table's privilege scheme.

For example, if you assigned a user an access level of 2, that user's access to the table, and to various fields within the table, are determined by the privileges you assigned to Level 2 in the table privilege scheme.

In building a table privilege scheme, note the following:

- A user's ability to access a table is a function of both the access level of the group and the user's individual access level. However, only the user's access level determines what the user can do with a table once it is opened.
- If you do not create a privilege scheme for a table, all users of the group can read and write to all fields in the table.
- Access rights cannot override a read-only attribute established for the table at the operating system level.

Table privileges

At the table level, you can control which operations each user access level (1–8) can do:

- View records in a table (read privilege)
- Change table record contents (update privilege)
- Append new records to a table (extend privilege)
- Delete records from a table (delete privilege)

When you create a table privilege scheme, all four table privileges are granted initially. That is, all table access levels are 1 by default (1 being the *least* restrictive level).

Field privileges

At the field level, you can control which operations each user access level (1–8) can do:

- Read and write to the field in the table (FULL privilege). This is the initial default.
- Read but not write to the field (READ ONLY privilege).
- Neither read nor write the field (NONE privilege). NONE blocks a user from writing to fields and from seeing fields you do not want to display.

About data encryption

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

When a DBF table's privilege scheme is saved, dBASE PLUS encrypts the table, including the production index (MDX) file and the memo (DBT) file, if any. dBASE PLUS also creates a backup copy of the original, unencrypted table. To ensure proper security, the backup files should be archived, then deleted from the system.

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

Planning your security system

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

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

1 Plan your user groups.

2 Plan each user's access level.

3 Plan each table's privilege scheme, including both table privileges and field privileges.

4 Implement your security scheme (see Setting up your DBF table security system).

Planning user groups

Take time to think through the various groupings into which you can divide your users, based on who needs access to which tables. For example, an administrative staff might need to access tables that a sales staff does not, or vice versa. Other groups may overlap; for example a marketing group might need to see some of the administrative tables and some of the sales tables.

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

Table 15.1 Setting user groups

Table	Group	User name
CUSTOMER	SALES	AMORRIS
		BBISSING
PRODUCT	ALL	AMORRIS

Planning user access levels

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

Although there are 8 access levels, you might choose to standardize on just 3 levels; one for full access, one for typical use, and one for minimal access. The next table shows the sample worksheet, expanded to show user access levels.

Table 15.2 Setting user access levels

Table	Group	User name	Level 1 (full access)	Level 4 (typical access)	Level 8 (minimal access)
CUSTOMER	SALES	AMORRIS	X		
		BBISSING		X	
		LJACUS	X		
		FFINE			X
PRODUCT	ALL	AMORRIS	X		
		BANDERS		X	
		BBISSING		X	
		CDORFFI		X	
		LJACUS	X		
		FFINE			X

Planning DBF table privileges

Next, plan each DBF table's privilege scheme.

For each table operation, determine the most restricted access level that can perform the operation. All levels less restricted than the specified one can perform that operation; all levels more restricted than the specified level cannot.

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

Table 15.3 Setting table privileges

Table	Read	Update	Extend	Delete
CUSTOMER	8	4	4	1
PRODUCT	8	4	4	1
ORDERS	8	4	4	1

Planning field privileges

The last planning step is to determine which user access levels can read and/or write to fields. Consider developing a worksheet similar to the following one.

Table 15.4 Setting field privileges

Field name	Full access	Read only	No access
PAYRATE	Levels 1–2	Levels 3–6	Levels 7–8
FIRSTNAME	Levels 1–6	Levels 7–8	
LASTNAME	Levels 1–6	Levels 7–8	

SSN	Levels 1–2	Levels 3–6	Levels 7–8

Setting up your DBF table security system

Once you've planned out your security scheme for DBF tables, you're ready to set it all up. Follow these steps to implement the security scheme:

1. In dBASE PLUS, define the database administrator password.
2. Define the user profiles, including group membership and access level.
3. Define table privileges.
4. Define field privileges.
5. Set the login security scheme.
6. Save the security information.

This section describes how to set the database administrator password, how to enter and edit user profiles, and how to set up table privilege schemes.

Defining the database administrator password

Before setting passwords, make sure any open tables have been closed. Follow these steps to enter the database administrator password:

1. Choose File | Database Administration. The Database Administration dialog box appears.
2. In the Database Administration dialog box, make sure that the Current Database field is set to <None> and the Table Type field is set for dBASE (DBF) tables.
3. Click the Security button. The Administrator Password dialog box appears.
4. In the Administrator Password dialog box, enter a password of up to 16 alphanumeric characters. You can enter characters in upper- or lowercase. The password does not appear onscreen.

The first time you set the administrator password you are prompted to reenter the password to confirm. (Thereafter, the system gives you three chances to enter the password correctly before the login terminates.) The Security dialog box appears.

Warning!

Once established, the security system can be changed only if the administrator password is supplied. Keep a hard copy of this password in a secure place. There is no way to retrieve this password from the system.

Creating user profiles

The Security Administrator dialog box is where you create user profiles and establish an access level for each user.

Follow these steps to add a user profile:

1. In the Security dialog box, select the Users tab and click the New button.
2. Enter a user login name (1–8 alphanumeric characters) in the User field. The entry is converted to uppercase. Required.
3. Enter a group name (1–8 alphanumeric characters) in the Group field. The entry is converted to uppercase. Required.
4. Enter a password for this user (1–16 alphanumeric characters). Required.
5. Select an access level for this user (from 1 through 8; see About groups and user access). Lower numbers give the greatest access; higher numbers are the most restricted.

6. Enter the user's full name (1–24 alphanumeric characters). This entry is optional. Because this item is not used in validating a login, you can use it any way you want. Frequently, the full name is used to add a more complete user identification. Alphabetic characters you enter in the Full Name option are not converted to uppercase.
7. Click OK to save the user profile.
8. The Security dialog box reappears with your new user info added to the list on the Users tab.
9. Repeat the preceding steps for each user.

Changing user profiles

To change a user's profile,

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

Warning!

Be careful when editing the group name, deleting the group, or deleting all users from a group. If you edit the group name, there is no way for its users to access tables associated with the original group. And if you delete the group or all users from a group before all tables associated with the group are copied out in a decrypted form, no one can access the tables. In that case, you must create a new user for the group.

Deleting user profiles

To delete a user profile,

1. Open the Users tab of the Security dialog box.
2. Select the user name of the user you want to delete, then click the Delete button.
3. To confirm the deletion, click the Yes button.

Establishing DBF table privileges

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

1. Open the Tables tab of the Security dialog box.
2. Select a table.
3. Assign the table to a group.
4. Establish the most restrictive access level for each table privilege.
5. Select field privileges for each user access level.

In general, for DBF tables you can use the Tables tab of the Security dialog box to

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

The sections that follow describe these steps in detail.

Selecting a table

To select a table,

1. Open the Tables tab of the Security dialog box. You use the Tables tab of the Security dialog box to create and modify DBF table privilege schemes. The DBF table privilege schemes are saved in the table structure.
2. In the Table field, type the name of the desired table. (Or click the Tools button and select the table.)
3. Click the Modify Table button. The Edit Table Privileges dialog box opens.

Assigning the table to a group

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

To select a group for the DBF table, click on the down arrow to display a list of the available groups from the Group list in the dialog box. (These groups were created when you created user profiles.)

Setting DBF table privileges

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

Table 15.5 Setting DBF table privileges

Privilege	Access granted
READ	View the table contents
UPDATE	Edit existing records in the table
EXTEND	Add records to the table
DELETE	Delete records from the table

To set table privileges, select a value (1–8) for each operation (Read, Update, Extend and Delete) in the dialog box. Remember that lower access numbers indicate the greatest access; higher numbers indicate the greatest restriction.

Note

You cannot specify access levels that are logically incompatible. For example, you cannot prohibit Level 6 from having read access, and also permit Level 6 to have update access. To have update access, Level 6 also needs read access.

Setting field privileges

With DBF tables you can establish access for each field by user access level. The following table describes the available field privileges.

Table 15.6 Setting field privileges

Privilege	Access granted
FULL	View and modify the field. This is the default.
READ-ONLY	View the field only (no update capability).
NONE	No access. The user can neither read nor update the field, and the field does not appear.

Note

Table privileges take precedence over field privileges. For example, if a table privilege is set for Read but not Update, the only meaningful field privileges are Read-Only or None. You must restrict *table* privileges to protect your data against table-oriented commands like DELETE and ZAP. Restricting field privileges to Read-Only or None without restricting table privileges doesn't protect data against these commands.

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

Follow this procedure to change a field privilege:

1. Select the field.
2. Click the Rights buttons that correspond to the privileges you want to grant for the field *for each access level*.
3. Repeat the process for each of the other fields in the table.
4. Click OK to save the field access privileges.

Warning!

Never change the access rights of the _DBASELOCK field of any table. The rights to this field must remain Full for all access levels.

Setting the security enforcement scheme

You can choose one of two enforcement schemes:

- Force a login when a user attempts to load dBASE PLUS itself.
- Force a login when a user tries to view a form linked to an encrypted DBF table. In this scheme, anyone may use unencrypted tables, but unauthorized users are prevented from accessing protected tables.

To change the security enforcement scheme, follow these steps:

1. Open the Enforcement tab of the Security dialog box. The two radio buttons on the Enforcement page indicate the security enforcement scheme currently in effect.
2. Select the enforcement scheme you want: whether to display a password dialog when loading dBASE PLUS or only when accessing an encrypted table.
3. Click Close.

Table-level security for DB tables

Although DB tables do not offer the extensive user the access-level and privilege- level security system available to DBF tables, DB tables (unlike DBF tables) support passwords.

You can use dBASE PLUS to assign master passwords to DB (Paradox) tables. Once you have assigned a master password assigned to a DB table, it cannot be opened without supplying the password, either by you locally or by users over the Internet.

You may choose to create a single master password that opens all DB tables. A user with this password need see only one password dialog to gain access to all DB tables. Or you may set unique passwords for particularly sensitive DB tables.

Note

In addition, auxiliary passwords are supported by DB tables, but you cannot access this feature from dBASE PLUS. Auxiliary passwords allow you to create multiple individual passwords for each DB table, so that you can restrict access to certain tables and certain fields. Different users can be given different passwords that will open only a

specific set of DB tables or allow read/write access to only certain fields within those tables. However, to set auxiliary passwords for field rights to a DB table you must use Paradox.

The process of assigning passwords is initially very similar to that described previously for DBF tables. To assign a master password to a DB table, follow these steps:

1. Make sure the DB table you want to secure is closed.
2. From the File menu, select Database Administration. The Database Administration dialog box appears.
3. Make sure that the Current Database field is set to <None> and the Table Type field is set for Paradox (DB) tables.
4. Click the Security button to open the Security dialog box.
5. Select the name of the table in the Table list. If the table is not in the current directory, use the Folder button to select the directory.
6. Click the Edit Table button to open the Master Password dialog box.
7. Enter the new password for the table in the Master Password field. The password can be up to 31 characters long and can contain spaces. Paradox passwords are case-sensitive.
8. Enter the password again in the Confirm password field.
9. Click the Set button to save the password.

Removing passwords from DB tables

To remove an existing password from a DB table, follow Steps 1 through 6 in the previous section. When prompted, enter the existing master password for the table. Then click the Delete button to remove the password from the table.

Chapter 16 Char sets language drivers

Chapter

16

Character sets and Language drivers

dBASE PLUS is an international software tool which can accommodate many languages. Each language or language category uses its own character set, and each has its own way of sorting and relating its characters. dBASE PLUS provides comprehensive language and character set support with multiple language drivers and automatic OEM–ANSI conversion as needed.

Users new to the Windows environment need to understand the difference between the OEM and ANSI character sets. Users who exchange data across national or linguistic boundaries need to understand how dBASE PLUS uses language drivers to handle data in different languages.

This section explains how dBASE PLUS uses the OEM and ANSI character sets and discusses techniques for working with language drivers.

Determining the language displayed by the User Interface

Language resources are files containing human-readable text strings that are displayed in the User Interface. In dBASE PLUS, these files are identified, according to language, by a two-letter code at the end of the file name. The online documentation (such as the Help files) are identified in the same way. The codes are:

- en = English
- de = German
- es = Spanish
- it = Italian
- ja = Japanese

While most users install dBASE PLUS for a single language, it is possible to select multiple languages during installation. You may also re-run the Installer at any time to install additional languages.

As with other dBASE PLUS settings, you may indicate a specific preference. If you do not indicate a specific preference, dBASE PLUS will follow the settings of the operating system (see Windows | Control Panel | Regional Settings). This is done on an as-available basis, dropping back to English when a target language resource cannot be

found. For example, if the Windows Regional Setting is set to German and the "de" (German language) version of a resource is available, that language resource will be used. A similar protocol is used for loading the Online Help.

You can indicate a specific dBASE PLUS language preference by making a selection from the Desktop Properties dialog. From the Desktop Properties | Country tab, you can access the User Interface Language picker. When you make a selection from the Country tab, your explicit preference will be written in the ERROR: Variable (ProductNameINI) is undefined. file. It will not take effect, however, until you re-start dBASE PLUS. Once restarted, dBASE PLUS will use the new language preference when possible.

At startup, locate the desired core-product language resource file. This file is identified as PLUS_xx.dll - where the "xx" is the two-letter language code. dBASE PLUS then attempts to load a language resource file by checking the following locations:

1. The ERROR: Variable (ProductNameINI) is undefined. file
2. The operating system's Regional Setting.

If a language resource file was not found at either location, dBASE PLUS will default to English and attempt to load any language resource file it finds. The language resource file that is successfully loaded determines the value of the _app. object's language property. The value of the _app.language property is the two-letter language code mentioned above.

The app.language setting will be used for the first attempt at locating relevant files when other language-specific resources and documentation files are needed. For example, when _app.language is set to "it", Italian, and a user invokes the Online Help system, dBASE PLUS will attempt to locate and load a file called "Plus_it.hlp". If this cannot be found, the system will attempt to load the file's English version.

About character sets

In the early 1980s, the developers of the IBM PC created an ordered list of symbols known as the *IBM extended character set*. This list contained all the classic ASCII 7-bit characters, together with various mathematical symbols, line and box drawing characters, and some accented characters.

While this was adequate for certain English-speaking countries, it was insufficient for most other countries. For example, there are accented characters in various European languages that are not included in the IBM extended character set. Therefore, a number of other character sets were developed. Each character set, including the original IBM extended character set, is contained in a *code page*. Each code page is designed for a particular country or group of countries, and each is identified by a three-digit number.

Some examples of the code pages supported by MS-DOS are:

- 437 English and some Western European languages
- 850 Most Western European languages
- 852 Many Eastern European languages
- 860 Portuguese
- 863 Canadian French
- 865 Nordic languages

These are known as *OEM code pages* (for Original Equipment Manufacturer). The classic IBM extended character set is contained in OEM code page 437 and is the default code page for the United States. DOS considers code page 850 to be the default for most European countries. Code page 850 contains all the letters (but not all the symbols) of code pages 437, 860, 863, and 865; consequently, many of the box-drawing and line-drawing characters contained in these code pages are omitted to make room for accented characters in code page 850.

Each character in a code page is identified with a number; this number (which can be decimal or hexadecimal) is known as a *code point*. For example, the code point of the numeric character "4" is 52 (decimal) or 34 (hexadecimal) in code pages 437 and 850.

The Windows environment uses its own character set, which is generally known as the *ANSI character set*. Although this character set shares many characters in common with the OEM code pages, it omits most of the line-drawing characters and mathematical symbols that these code pages offer. Furthermore, even characters shared in common between an OEM code page and the ANSI character set often have different code point numbers.

The global language driver determines the character set used by dBASE PLUS. If you have another product already installed on your system that uses the Borland Database Engine (BDE), your current language driver is unchanged when you install dBASE PLUS. If no BDE language driver setting is detected, however, dBASE PLUS installs the ANSI language driver by default.

About language drivers

dBASE PLUS uses language drivers to specify which character set to use and which language rules apply to that character set. For example, the Canadian French language driver uses a character set that is identical to code page 863, while the default driver for the United States uses a character set that is identical to code page 437. It is important to understand that dBASE PLUS uses these internal code pages instead of the code pages supplied by the operating system.

dBASE PLUS language drivers contain tables that define or control the following for a particular character set:

- Alphabetic characters
- Rules for upper- and lowercase
- Collation (sort order) used in sorting or indexing
- String comparisons (=, <, >, <=, >=)
- Soundex values (values that represent phonetic matches when exact spellings are not known)
- Rules for translation between OEM and ANSI character sets

dBASE PLUS identifies each driver with a character string known as an *internal name*. For example, the internal name of the German driver for code page 850 is DB850DE0, and the internal name of the Finnish language driver is DB437FI0. The following table lists some of the European language drivers available in dBASE PLUS.

Table 16.1 European language drivers available in *dBASE PLUS*

Language or country	Code page	Internal name
Portuguese/Brazil	850	DB850PT0
Portuguese/Portugal	860	DB860PT0
Danish	865	DB865DA0
Finnish	437	DB437FI0
French/Canada	850	DB850CF0
French/Canada	863	DB863CF1
German	437	DB437DE0
Italian	437	DB437IT0
Netherlands	437	DB437NL0
Norway	865	DB865NO0
Spanish	437	DB437ES1
Spanish	ANSI	DBWINES0

Swedish	437	DB437SV0
English/UK	437	DB437UK0
English/UK	850	DB850UK0
English/USA	437	DB437US0
English/USA	ANSI	DBWINUS0
W. European	ANSI	DBWINWE0

When dBASE PLUS converts data from OEM to ANSI, and vice versa, most alphabetic characters exist in both an OEM code page and the ANSI character set and are converted without problem. Most of the extended graphic symbols in an OEM code page cannot be represented in the ANSI character set at all. When such a discrepancy exists, dBASE PLUS, like other standard Windows applications, makes a guess at the nearest character, but data loss can occur.

Performing exact and inexact matches

When dBASE PLUS compares two characters, either an exact match (also known as a *primary match*), or an inexact match (also known as a *secondary match*) can be performed. An exact match is performed when SET EXACT is set to ON, and in inexact match is performed if EXACT is set OFF.

An exact match requires that the characters be exactly the same. For example, although the characters O and Ö are similar, they don't satisfy the requirement for an exact match, and a SEEK or a FIND expression treats them as different characters. In contrast, an inexact match requires only that the characters belong in the same general category. For example, since the characters O and Ö are similar in several languages, they satisfy the requirement for an inexact match with many language drivers; a SEEK or a FIND expression treats them as identical characters.

Exact and inexact matches are performed using *primary weights* and *secondary weights*, which are assigned by a language driver to each character. Exact matches use primary and secondary weights, while inexact matches use only the primary weights and ignore the secondary weights. For example, the SET EXACT command controls whether characters with umlauts match their respective characters without umlauts. The following commands open a table named VOLK.DBF and search for a record with a key value that satisfies an exact match criterion:

```
set exact on
use VOLK order NAMEN
seek "KONIG"          // Will NOT find "KÖNIG".
```

EXACT is ON and the secondary weights of O and Ö are different, so they are evaluated as different characters. The following commands open VOLK.DBF and search for a record with a key value that satisfies an inexact match criterion:

```
set exact off
use VOLK order NAMEN
seek "KONIG"          // Can find "KÖNIG".
```

EXACT is OFF and the primary weights of O and ö are the same, so they are evaluated as identical characters.

Using global language drivers

Each time you start dBASE PLUS, a language driver is activated automatically. This is known as the *global language driver*. This setting applies to reading and writing of files, table creation, table-independent character operations and the output of commands and functions. For example, the global language driver governs FOR and WHILE expression evaluations.

dBASE PLUS normally chooses the global language driver from the dBASE Language Driver setting in the BDEADMIN.EXE Utility. Optionally, you can also specify a global driver in your ERROR: Variable (ProductNameINI) is undefined. file with an *ldriver* key. When there is no ERROR: Variable (ProductNameINI) is

undefined. entry for a language driver, the setting in the BDEADMIN Utility determines the global language driver. When you place a valid driver entry in ERROR: Variable (ProductNameINI) is undefined. it overrides the setting in the BDEADMIN Utility except when creating tables. dBASE PLUS will always set the new table's language according to the global language driver specified in the BDE Administrator Utility.

To set the ldriver option in ERROR: Variable (ProductNameINI) is undefined.:

1 Close dBASE PLUS if it is running.

2 Open the ERROR: Variable (ProductNameINI) is undefined. file (normally located in your Plus\BIN directory) and enter one of the following in the [CommandSettings] section:

ldriver = WINDOWS

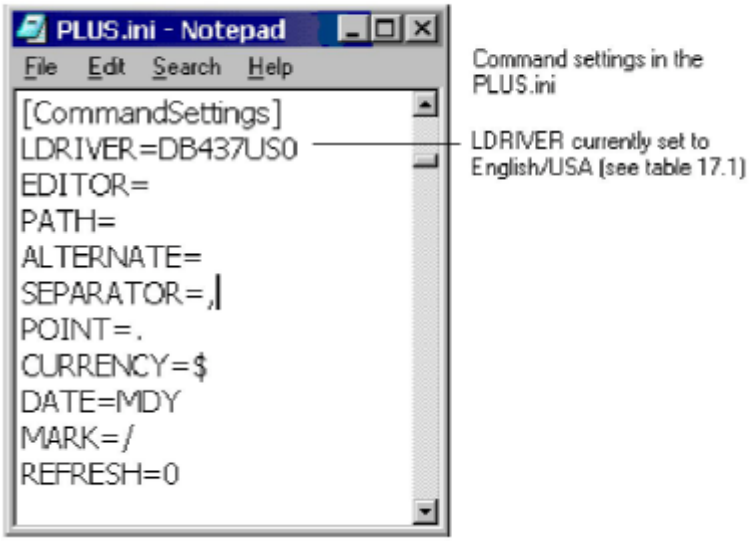
or

ldriver = <internal driver name>

For example, the internal name of a European Spanish language driver for code page 437 is DB437ES1; to install this driver, insert the following setting:

ldriver = DB437ES1

Figure 0.1 Setting LDRIVER in the PLUS.ini



3 Save your changes and restart dBASE PLUS.

Use ldriver = WINDOWS to maximize compatibility with the operating system locale.

Use ldriver = <internal driver name> to specify a Borland language driver and maximize compatibility with legacy applications. For legacy applications matching the global language driver to the one previously in effect will help ensure compatible character handling and processing of data in the legacy tables.

Using table language drivers

dBASE PLUS assigns a language driver to a table automatically when you create it. This assignment is recorded in the LDID, a 1-byte identifier in the file header region. When you create a table from scratch, dBASE PLUS always assigns the current global language driver to the LDID. When you create a table file from another table file, either the global language driver or the language driver of the original table is assigned to the LDID of the new table.

Which language driver is assigned depends on the command you use to create the file, as shown in the following table:

Table 16.2 Automatic assignment of language drivers by *dBASE PLUS*

Assigns global driver to the LDID of new table	Assigns original table driver to LDID of new table
CREATE	COPY FILE
CREATE...FROM	COPY STRUCTURE
CREATE...STRUCTURE EXTENDED	COPY...STRUCTURE EXTENDED
	COPY TABLE

For example, the following commands open a table file and create a new one with the LDID set to the current global language driver:

```
use CLIENTS     // LDID specifies a language driver other than global language driver
copy to CLIENTS2 structure extended  // LDID of CLIENTS2.DBF matches the language
                        // driver of CLIENTS.DBF
use CLIENTS2 exclusive
create NEWCLIENT from CLIENTS2       // Create a new table with the global LDID
```

The following commands open a table file and create a sorted table file with an LDID set to the original table language driver:

```
use CLIENTS                 // LDID specifies a nonglobal language driver
sort on LASTNAME to CUSTOMER  // LDID of CUSTOMER the same driver as with CLIENTS
```

Identifying a table language driver and code page

Because some commands behave differently than others when a table language driver differs from the current global language driver, it is often necessary to detect which language driver is assigned to the LDID region of the table file or determine which code page the language driver uses. For example, file and field names may be valid in one language but not in another, or a key field may have characters that are not shared in common between the code pages of the language drivers.

When you use a command to open a table, and that table has a language driver that differs from the global language driver dBASE PLUS displays a warning dialog box only if LDCHECK is set to ON (installation default is OFF).

To determine which language driver is recorded in a table LDID region and which code page the driver uses, use the LDRIVER( ) and CHARSET( ) functions:

```
set ldcheck off      // Turns off automatic language driver compatibility checking.
use CLIENTS exclusive
index on COMPANY tag COMPANY
if ldriver( ) == "DB437DE0"   // If this is the German language driver...
  seek "Shönberg Systems"                 // Searches are OK
else
  if charset( ) == "DOS:437" // If the driver uses Code Page 437...
    warn1.open( )          // Opens a custom warning dialog box.
  else                   // If the driver doesn't use Code Page 437...
    warn2.open( )        // Opens a different warning dialog box.
  endif
endif
```

Non-English Character Display Issues

There are two parts to this issue:

- First, we must make sure dBASE PLUS uses the correct code page when interpreting text encoded in source files (.prg, .wfm, etc.), or text encoded in an incoming byte stream from a dBASE PLUS Web App.
- Second, we must make sure dBASE PLUS uses a display font that contains the characters we wish to display. For font specification, see Selecting Specialized Product Fonts.

dBASE PLUS interprets text according to the code page associated with the language driver you specify. You can set the language driver in the ERROR: Variable (ProductNameINI) is undefined. file through the CommandSettings section as follows:

```
[CommandSettings]
ldriver=<internal Name>
```

For a complete list of Language Drivers and their internal names, please see the topic "About language drivers". Select the language driver that best matches the language you wish to display.

In addition to using the ldriver setting, you can also use the BDE to designate a desired language driver. In the absence of an ldriver setting in the ERROR: Variable (ProductNameINI) is undefined., dBASE PLUS uses the default language driver from the BDE configuration. This BDE setting can be used to designate alternative language drivers in much the same fashion as an ldriver setting. Please note that an ldriver setting takes precedence and will therefore override any subsequent change to the BDE configuration.

Selecting Specialized Product Fonts

In order to use TrueType fonts which do no use the Western Europe code page (1252), you must specify the language (also referred to as the "script"). Since dBASE PLUS does not list available language scripts for TrueType fonts, you must specify it in the *fontName* property--either in code or through The Inspector--using the exact TrueType font name. If you use The Inspector, choose a text component, it's *fontName* Property and, instead of choosing from the fonts list, type in the name of a desired language script. We recommend all languages be entered in English, e.g.:

- Times New Roman Greek
- Verdana Turkish
- Arial Baltic
- MS Gothic Cyrillic
- Courier New Central Europe

The following ERROR: Variable (ProductNameINI) is undefined. file settings ensure that the initial font created for a new control uses the language you want:

```
[DefaultFonts]
Application=<strFontName>,<intPointSize>
Controls=<strFontName>,<intPointSize>
```

The Application setting specifies the font used for the Navigator and Inspector, while the Controls setting specifies the default font used for forms and controls. You can also create your own custom controls to specify the font and language you want to use.

Table language drivers versus global language drivers

When the language driver of a table differs from the current global language driver, the table language driver is loaded into memory automatically when you open the table. Thereafter, the table language driver is respected by some commands, while the global language driver is respected by others.

All commands that have nothing to do with a table use the global language drivers. The following table shows the general rules when operations are performed on table data where the table language driver differs from the global language driver.

Table 16.3 Language drivers: Table versus Global

Table driver	Global driver
INDEX ON command expressions	SET FILTER command expression
FOR scope expression of INDEX ON command	FOR and WHILE expressions for every command except INDEX ON
SET KEY range checking expression	SET RELATION TO expression
SORT command expressions	
Secondary matches for expressions in LOOKUP( ), FIND, SEEK, and SEEK( ) with EXACT set OFF	
Secondary matches for SET RELATION TO expression with EXACT set OFF (uses the driver of the child table)	

For example, when you create a table file with the German language driver, an LDID identifier is written to the header region of the file. If the global language driver is set to English and you open the table in dBASE PLUS, dBASE PLUS notes the discrepancy between the table's and system's language rules. If you create an index with INDEX ON, the logical order of the index obeys the language driver of the table:

```
use VOLK                  // Created with the German language driver
index on NAMEN tag DIENAMEN  // Orders records in the German way
```

By contrast, if you create a filter with SET FILTER, the filtering condition obeys the global language driver:

```
use VOLK
set filter to NAMEN = "KONIG"  // Excludes records with "KÖNIG" in NAMEN
```

Handling character incompatibilities in field names

When you use a table file that was created with a different language driver from the current global language driver, some characters in the table file might not be recognized. This can lead to problems.

For example, the German language driver DB437DE0 has the same code page and character set as the US language driver DB437US0. However, the German language driver recognizes extended characters like ö and Ü as alphabetic characters, while the U.S. language driver doesn't. Consequently, when a field name contains such characters (as with ÜNAMEN in the example below) problems arise when you try to reference the field in when using the U.S. language driver.

When such conditions exist, the following command generates an error condition:

```
replace ÜNAMEN with "Schiller"
```

You can solve this problem by surrounding the field name with the : delimiter, which treats the field name as an identifier regardless of the rules contained in the current language driver:

```
replace :ÜNAMEN: with "Schiller"
```

When you use this command, each element in the field name ÜNAMEN, including Ü, is treated as an identifier and the command executes successfully.

Converting between OEM and ANSI Text

The Source Editor Properties dialog box lets you specify whether to view the text, in an editor, in the DOS (also called OEM or ASCII) character set or the Windows (also called ANSI) character set. Only the way you view the source changes, not the actual code points.

You can also convert between character sets. For example, you may want to convert from the OEM character set to the ANSI character set if you are changing to an ANSI language driver, and your program uses extended characters.

Warning!

Converting your program to a different character set changes the code points of values in your program. You may lose information if a character you convert does not exist in both character sets.

You should carefully review extended characters in your code if you convert between character sets.

Converting from OEM to ANSI

The encoding is changed when you convert between character sets. For example, if your program was written in the OEM character set and it uses the Ä character (OEM 142), the actual numeric value of the character is changed to 0196 when you convert the program to ANSI, because 0196 is the numeric value of Ä in the ANSI character set.

Alphanumeric characters usually do not cause problems when you convert from an OEM character set to the ANSI character set. Characters that typically do not convert include:

- Greek characters other than ß (ANSI 0223) and µ (ANSI 0181)
- Line-drawing and box characters

Converting from ANSI to OEM

Lowercase alphanumeric characters in the ANSI character set exist in all OEM character sets and are converted without problems. Uppercase extended alphabetic characters exist in some OEM character sets (such as OEM code page 850) but not in others (such as OEM code page 437). Following are the rules dBASE PLUS uses for conversion:

- Extended characters in the ANSI set that do not exist in the OEM character set are converted to "similar" characters. For example, the accented À (ANSI 0192) is converted to the capital A in OEM code page 437.

How to convert and view your source code

To convert between character sets:

1. Open your program or text file in the editor.
2. Select the section you want to convert or Select | All.
3. Choose Edit | Convert | To DOS Text to convert your source to OEM, or Edit | Convert | To Windows Text to convert your source to ANSI.

Note

Converting between character sets does not change the character set you are using to view your text or program.

To change the way you view your text or program:

1. Choose Properties | Source Editor Properties.
2. Specify either DOS Text or Windows Text for Interpret text as.

Chapter 17 Making ADO Connections

Chapter

17

Making an ADO Connection

Setting up an ODBC / ADO Driver

Steps needed to load an ODBC driver to use with ADO support in dBASE PLUS 9. This paper assumes the user will be using Windows 7 or 8 – 64 bit operating systems and is detailing how to set the environment with MySQL drivers.

Microsoft Windows Database Connectivity support

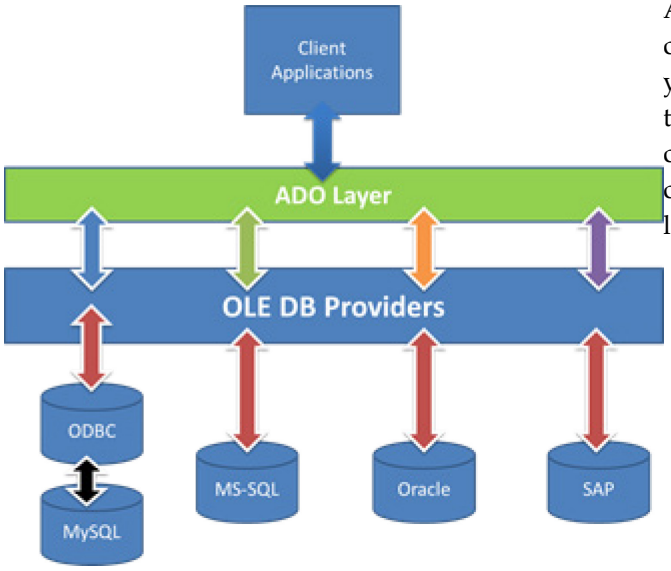
If you are not familiar with ADO from Microsoft, the following is from Microsoft:

"Microsoft ActiveX Data Objects (ADO) enable your client applications to access and manipulate data from a variety of sources through an OLE DB provider. Its primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. ADO supports key features for building client/server and Web-based applications." Microsoft – MSDN

ADO, much like the BDE, also supports the other database connectivity solution called ODBC (Open Database Connectivity) and it will allow connection to be passed from ODBC through ADO, again much like the BDE.

How to connect using ADO

ADO uses OLE DB providers for all connections. You can have a OLE DB provider specifically for your database or you can use an ODBC driver and use the Microsoft 'MSDASQL.1' OLE DB provider to connect to it.



ADO uses an OLE DB provider for ODBC connections if all you have is an ODBC driver for your database. However, some applications have their own OLE DB providers that allow you to connect more directly to the database. The ADO connection path is layed out in the image on the left.

1 – Connect using ODBC Driver.

There are two ways to do this. You can create an ODBC User DSN or you can just use the Connection String property to set it up.

A: Using a DSN

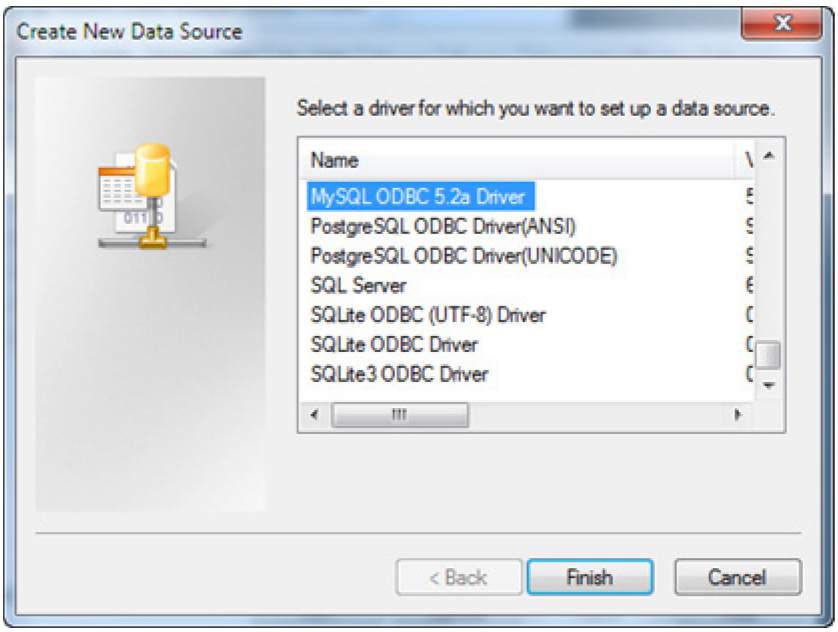
Create the DSN first. Then you can either use a .udl file or just write the ConnectionString yourself.

Here is an example creating an ODBC User DSN for a MySQL database. Then using a .udl to create the connection string.

First Create the User DSN …

Open the ODBC Administrator (dBASE includes a link to the 32bit ODBC Administrator here : ALL PROGRAMS | dBASE PLUS 9| 'ODBC Administrator (32bit)

Once the ODBC Administrator is open .. create your DSN.
Click 'Add'. A dialog similar to the one shown here will open.
Choose your Driver and click 'Finish'.



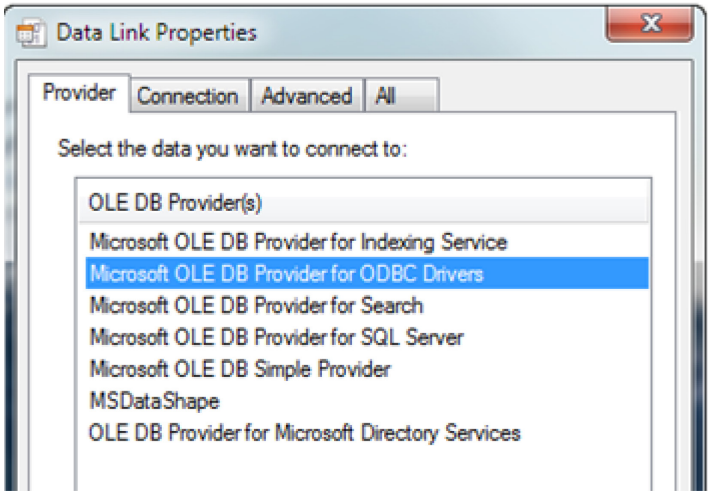
A Dialog will pop up that is specific to the Database. In this case I am connecting to a MySQLdatabase …

Once your DSN is set up ….

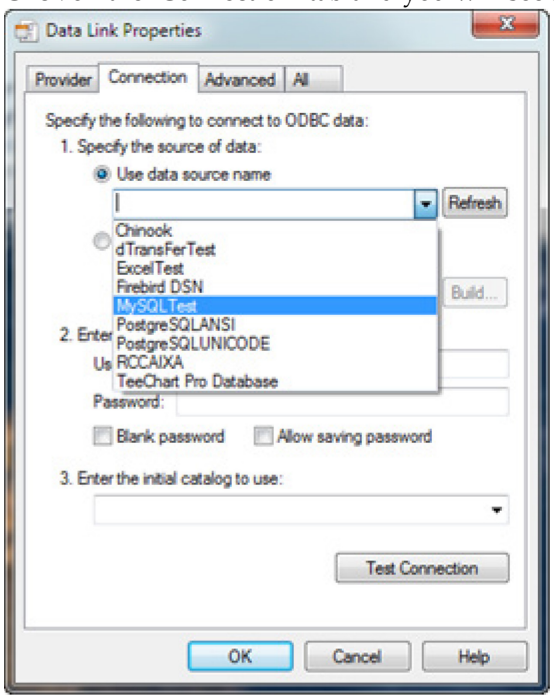
Using the UDL to create an ADO Connection String …

In Windows Explorer, find the folder you want to use to create the UDL … Right click and choose 'New' … 'Text Document'. Name the new document something with a udl extension … TestMySQLConnect.udl (Say 'Yes' when asked if you want to change the name).

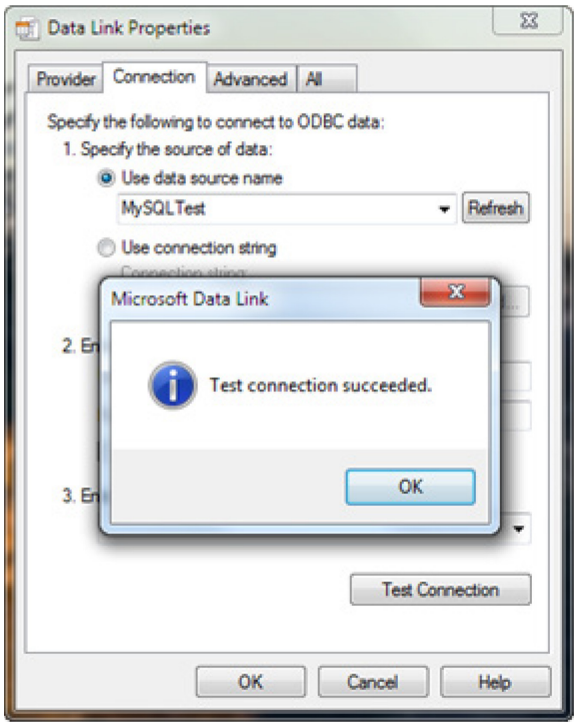
Double click the new .udl and you will see the Microsoft OLE DB Providers dialog. In this case we will use the 'Microsoft OLE DB Provider for ODBC Drivers' …



Click on the 'Connection' tab and you will see something like this…



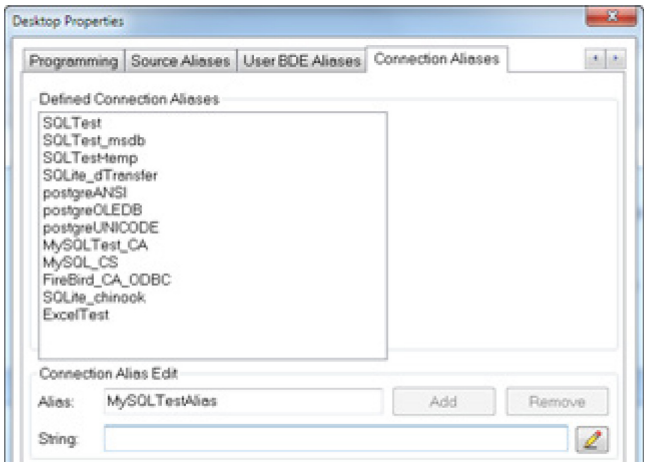
Since the User Name and Password and other information was provided in the DSN .. we do not need to do it here. Just choose the DSN and click 'Test Connection' to test.



Now in dBASE all we need to do is use this udl to create the connection string in dBASE.

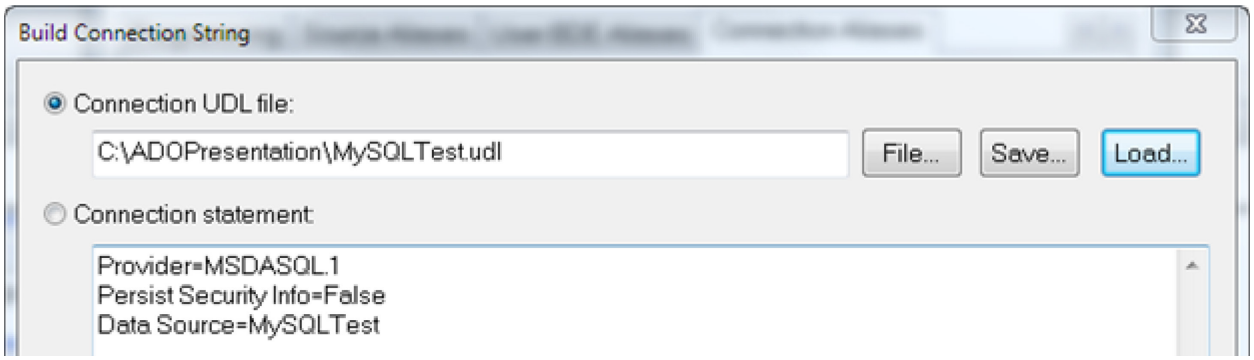
Creating the Connection String …

In the dBASE Menu go to Properties | Desktop Properties .. to get the Desktop Properties dialog. Go to the 'Connection Aliases' tab and you will see somehting like this …



Where it says 'Alias:' add a new Connection Alias. Here I will use 'MySQLTestAlias' … then click on the yellow pencil to open the Connection String Dialog.
(*IMPORTANT NOTE: If you are using a DSN … it cannot be the same name as the Connection Alias*)

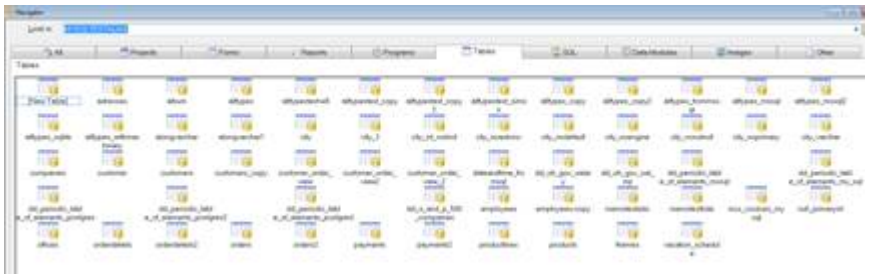
Choose 'Connection UDL File' … and click on 'File…'. Find the .udl file you just created and add it. Then click on the 'Load' button and you will see something like this ..



You connection statement (AKA Connectioln String) is "Provider=MSDASQL.1;Persist Security Info=False;Data Source=MySQLTest".
Click OK … You will return to the Connection Alias dialog. Click 'Add' to make sure it is added to the list.

(*IMPORTANT NOTE: you can bypass the .udl and just put your connection statement here by writing it by hand. If you are using a DSN. Most likely the Connection Statement is always going to be "Provider=MSDASQL.1; Persist Security Info=False; Data Source=<yourDSN>". You don't even have to open the 'Build Connection String' dialog. You can just put this code, with semi colons separating the elements of the statement, directly in the 'String' entryfield under the Connection Aliases tab.*)

Now you are ready to use your ADO connection to your database. You can check this by going in to the Navigator … click on the 'Tables' tab…. Choose the ADO Connection Alias you just created … and you will see a list of the tables in that database.



B: Bypassing the DSN and using the Connection String only

Now were going to Just create a connection string using a connection string without having to create a User DSN.

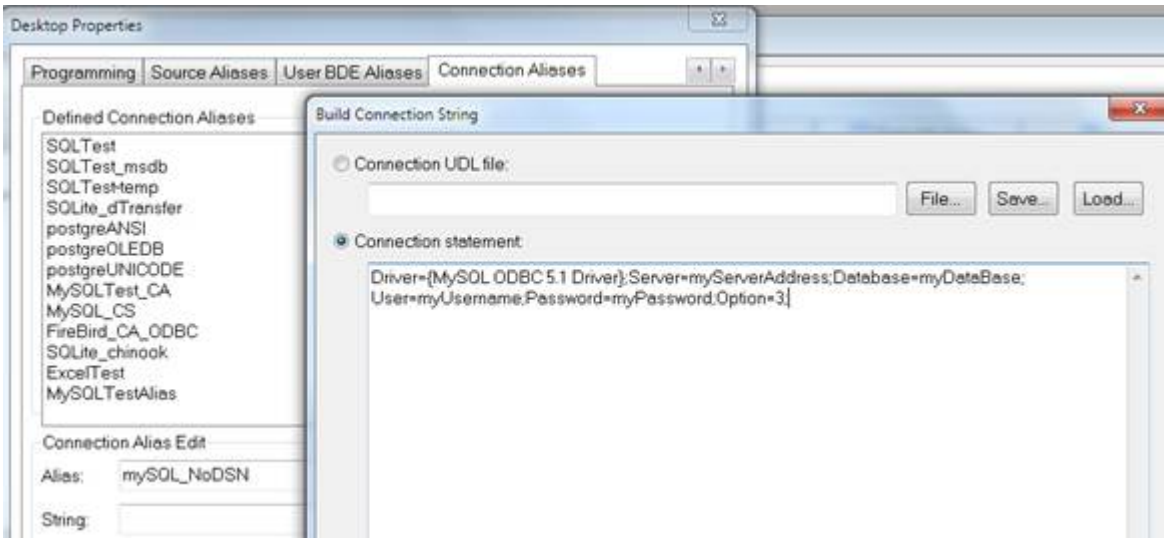
First you need to know what kind of connection string to build. There are many excellent sites for this, one of which is http://www.connectionstrings.com/ Here I did a search for a MySQL 5.1 using ODBC and connecting to a remote server. This is one of the samples they had…

Remote database

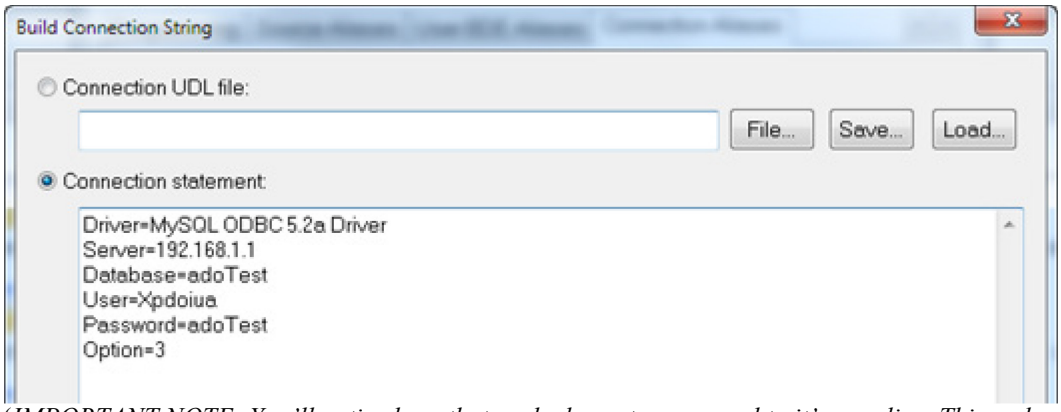
```
Driver={MySQL ODBC 5.1 Driver}; Server=myServerAddress; Database=myDataBase;
User=myUsername; Password=myPassword; Option=3;
```

MySQL

You can take this string copy it and in the build string dialog of the new Connection Alias



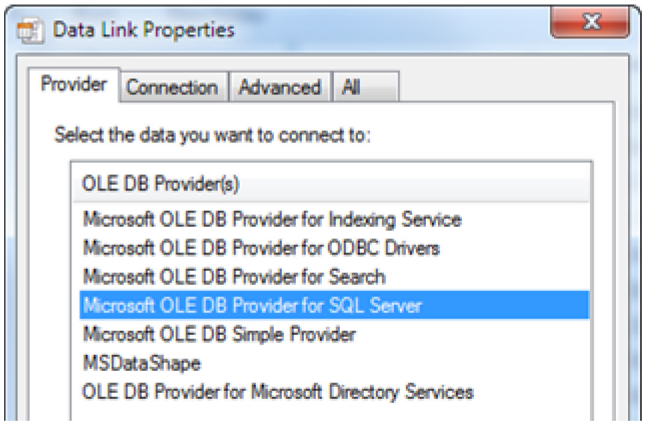
you can copy this string and simply plug in the values for your database.



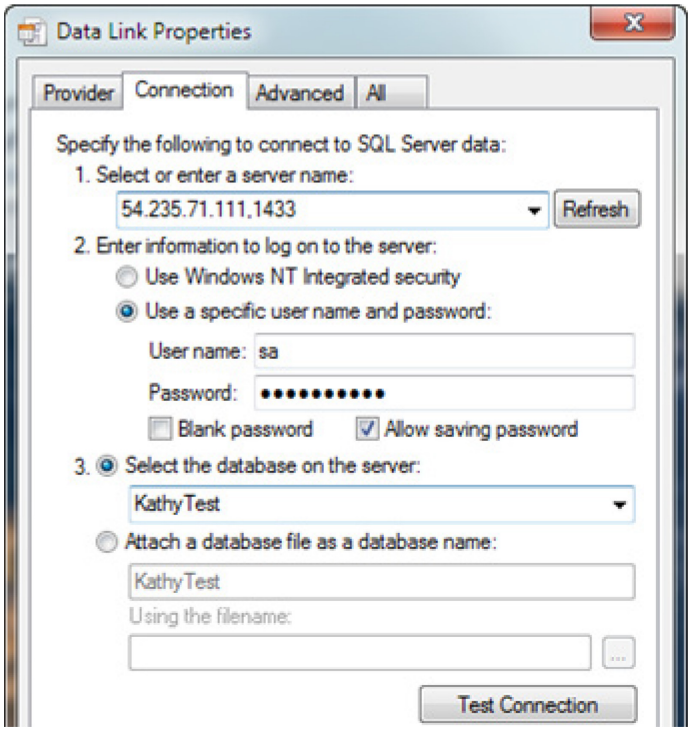
(*IMPORTANT NOTE: You'll notice here that each element was moved to it's own line. This makes viewing and editing the connection string much easier and it doesn't make the connection string invalid*)

2 – Connect using OLE DB Provider Only.

You may have an OLE DB Provider for your database. Again, you can create your own connection string or you can use a udl file. Here is an example using the .udl file to connect to a SQLServer Database using it's OLE DB provider. Create a new UDL file (see above). Choose the OLE DB Provider..



Under Connection .. enter the server info to connect to the database …

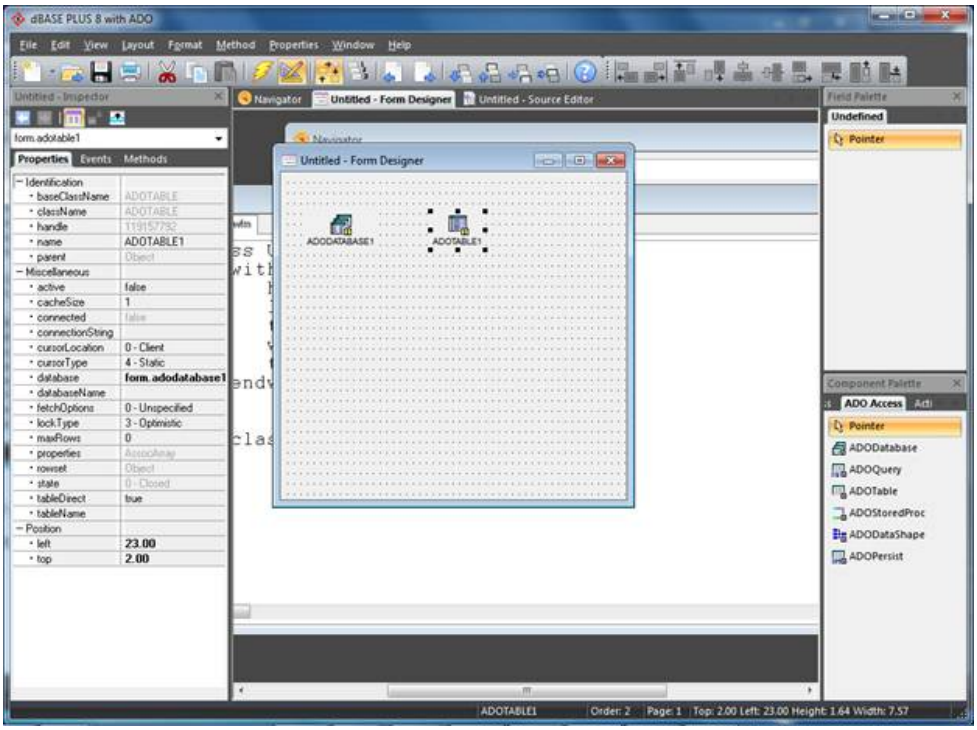


Test your connection and once you have it set up correctly, you can use the .udl to create the connection string (and Connection Alias) in dBASE (See instructions above under "**Creating the Connection String … "**) . If you don't want to use the .udl dialog you can also create your own connection string by hand (see the above section on "**Bypassing the DSN and using the Connection String only**")

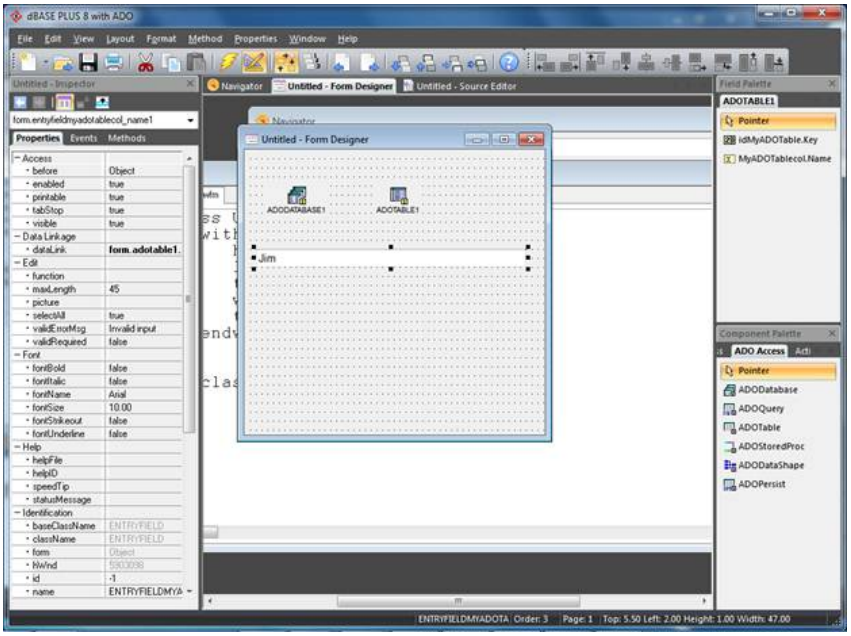
Using the Connection in dBASE PLUS 9:

Now it is time to connect the defined databases with the included ADO components.

Create a new form, but do not use the Wizard. Now go to the ADO Access tab on the Component palette. Drop an ADODataBase and an ADOTable component onto the form.



In the Fields tool window, drag a Field to the form.



You should see the Field on your form showing data.