Contents

Index of all included TIs

File	Ext	Size	Date	Time	Description
ti2529	asc	7690	02/28/96	009:53	Uninstalling Delphi (16 bit)
ti2558	asc	4954	02/28/96	009:53	Installing Delphi on a Network
ti2602	asc	1130	02/28/96	009:53	Making a form selectable without the main form.
ti2616	asc	728	02/28/96	009:53	How to get a file's date and time stamp.
ti2617	asc	2355	02/28/96	009:53	Lists the virtual key values
ti2619	asc	1374	02/28/96	009:53	How to paint the form with a bitmap.
ti2621	asc	3432	02/28/96	009:53	How to associate a string with a component.
ti2622	asc	1803	02/28/96	009:53	How close a file opened from a Delphi DLL in VB.
ti2645	asc	1354	02/28/96	009:54	How to readln longer than 255 chars.
ti2647	asc	1455	02/28/96	009:54	A VERY short primer on dynamic memory allocation.
ti2654	asc	1462	02/28/96	009:54	How to use array of const.
ti2656	asc	28722	08/05/96	2:01p	Function mapping from the Paradox Engine to BDE
ti2659	asc	715	02/28/96	009:54	Making your own hotkeys.
ti2661	asc	1180	02/28/96	009:54	How to scroll your form with pgUP and pgDn.
ti2683	asc	2298	02/28/96	009:54	Drag a form without clicking the caption bar
ti2695	asc	7849	02/28/96	009:54	New Language Features in Delphi 2.0 - 32 Bit
ti2711	asc	3309	12/13/95	3:37p	How to Circumvent the "index not found" Exception.
ti2719	asc	4564	02/28/96	009:54	College Student Guide to Reading and Writing Files
ti2751	asc	1268	03/12/97	4:01p	Some current internal limits of BDE
ti2751	asc	2349	04/02/97	4:59p	Some current internal limits of BDE
ti2751	asc	1091	08/05/96	2:04p	Some current internal limits of IDAPI
Some current internal limits of BDE					
ti2752	asc	2525	08/05/96	2:05p	Local SQL Reserved Words.
ti2761	asc	22071	08/05/96	2:05p	Introduction to BDE Programming
ti2762	asc	8285	08/05/96	2:06p	Comparison filters.
ti2763	asc	11534	08/05/96	2:07p	Using a continue node in a filter.
ti2767	asc	18281	02/28/96	009:54	Delphi Fact Sheet
ti2768	asc	7882	11/05/96	11:31a	Delphi Quick Info Sheet
ti2770	asc	16424	01/23/97	4:04p	BDE Frequently Asked Questions.
ti2776	asc	4522	02/28/96	009:55	List of Delphi Books From Third-Party Publishers
ti2777	asc	2167	02/28/96	009:55	Instructions for Running Delphi from CD-ROM
ti2778	asc	1645	11/13/96	008:07	Making your Delphi apps show minimized.
ti2779	asc	39024	02/28/96	009:55	Delphi Client/Server and Power Builder Compared
ti2780	asc	33015	02/28/96	009:55	Delphi vs Visual Basic

ti2781 asc 3525 02/28/96 009:55 Step by Step Configuration of an ODBC Driver
ti2783 asc 7141 02/28/96 009:55 Creating Database Aliases in Code
ti2790 asc 3638 06/24/96 4:03p Creating & Deleting TFields at run-time
ti2791 asc 3671 02/28/96 009:55 How to iterate through the fields of a table
ti2792 asc 1242 02/28/96 009:55 How to match file date/time stamps.
ti2793 asc 6303 02/28/96 009:55 Adding Graphics in Your Listboxes and Comboboxes
ti2796 asc 1673 02/28/96 009:56 Using OnHint Events Among Mulitiple Forms
ti2798 asc 1636 02/28/96 009:56 Making the Enter key work like a Tab in a TDBGrid
ti2799 asc 994 02/28/96 009:56 Moving to a tab by name on a TabSet
ti2800 asc 1131 02/28/96 009:56 Calling a Delphi DLL from C
ti2801 asc 1127 02/28/96 009:56 Getting a query's Memo field as a string
ti2803 asc 1292 02/28/96 009:56 How to encrypt a String
ti2804 asc 1010 02/28/96 009:56 Undo in a Memo field
ti2805 asc 837 02/28/96 009:56 Getting the Line number in a memo Field
ti2807 asc 4528 02/28/96 009:56 Loading Bitmaps Into dBASE And Paradox BLOB Fields
ti2809 asc 1373 02/28/96 009:56 Printing the contents of a TMemo or TListbox
ti2810 asc 2193 02/28/96 009:56 Extracting A Bitmap From A BLOB Field
ti2811 asc 9000 02/28/96 009:56 Bitmaps And InterBase BLOB Fields
ti2812 asc 1220 02/28/96 009:56 Control Font Styles
ti2814 asc 5584 02/28/96 009:56 Handling EDBEngineError Exceptions
ti2815 asc 1142 02/28/96 009:57 How to handle text drawing in a TDBGrid
ti2816 asc 1809 02/28/96 009:57 Trouble running Delphi programs from Delphi
ti2817 asc 2529 08/05/96 2:09p Understanding the PARADOX.NET file with the BDE
ti2818 asc 1123 02/28/96 009:57 Searching your application's help file
ti2820 asc 7824 02/28/96 009:57 Searching Through Query Result Sets
ti2821 asc 18965 02/28/96 009:57 dBASE .DBF File Structure
ti2822 asc 3165 08/05/96 2:11p Setting File Handles For A Windows BDE Application
ti2824 asc 1880 02/28/96 009:57 Passing a Variable to a ReportSmith Report
ti2825 asc 8640 02/28/96 009:57 Extracting Index Data From A Table
ti2834 asc 2993 02/28/96 009:57 Redistributing the Borland Database Engine
ti2837 asc 5740 02/28/96 009:57 Cascading Deletes With Pdox Referential Integrity
ti2838 asc 17091 02/28/96 009:57 dBASE Expression Indexes: A Primer
ti2840 asc 1388 02/28/96 009:57 Removing the vertical scrollbar from a TDBGrid
ti2841 asc 1475 02/28/96 009:58 Delphi Consultants and Training Centers
ti2842 asc 5441 02/28/96 009:58 InterBase BLOB Fields: A Primer
ti2844 asc 9985 02/28/96 009:58 Using The ASCII Driver With Comma-delimited Files
ti2849 asc 5669 02/28/96 009:58 Determining Record Number In A dBASE

Table
ti2854 asc 7288 02/28/96 009:58 Managing disk volume labels in Delphi
ti2855 asc 2249 02/28/96 009:58 How to copy files in Delphi.
ti2856 asc 1184 02/28/96 009:58 How to terminate all running applications
ti2857 asc 1326 02/28/96 009:58 How to check to see if a drive is ready.
ti2858 asc 2716 02/28/96 009:58 How to do pointer arithmetic in Delphi.
ti2859 asc 1432 02/28/96 009:58 How to do bit-wise manipulation.
ti2860 asc 2058 05/21/96 008:30 Basic Delphi DLL template
ti2861 asc 6567 02/28/96 009:58 Form display with different screen resolutions.
ti2862 asc 924 02/28/96 009:58 How to keep the app iconized.
ti2863 asc 1064 02/28/96 009:58 How to use a custom cursor.
ti2864 asc 1323 02/28/96 009:59 How to use a popup menu with a VBX.
ti2865 asc 1556 02/28/96 009:59 How to set a max and min form size.
ti2866 asc 1328 02/28/96 009:59 How to get the windows and DOS versions.
ti2867 asc 1073 02/28/96 009:59 How to tell what kind of drive is used.
ti2869 asc 2503 02/28/96 009:59 How to determine the current record number.
ti2870 asc 1098 03/01/96 10:23a How to automate logon for Paradox tables
ti2873 asc 1051 05/21/96 008:30 packing a dBASE table
ti2892 asc 2424 05/21/96 008:30 string manipulation routines
ti2894 asc 1148 02/28/96 009:59 WinExecAndWait
ti2895 asc 1795 05/21/96 008:30 How to check for app already running.
ti2896 asc 1614 06/14/96 4:06p How to use a form several times
ti2899 asc 2212 02/28/96 009:59 Manually Installing Delphi
ti2901 asc 1463 06/14/96 4:07p Printing in the DOS IDE under Windows 95
ti2903 asc 1420 06/14/96 4:07p Different colored characters in a string grid
ti2906 asc 2576 02/28/96 009:59 Returns the amount required to repay a debt.
ti2909 asc 895 02/28/96 009:59 How to click and move components at runtime.
ti2919 asc 1703 08/05/96 2:12p Changing the NET DIR Programmatically
ti2936 asc 7586 03/05/96 0:54p Delphi 1.02 Maintenance Release Information
ti2938 asc 5050 06/24/96 4:04p Creating Dynamic Components at Runtime
ti2945 asc 1076 03/04/96 009:54 Loading a Custom Cursor from a RES File
ti2947 asc 4742 03/05/96 0:54p Loading Bitmaps and Cursors from RES Files
ti2948 asc 3651 03/04/96 009:54 SQL: Embedded Spaces in Field/Column Names
ti2949 asc 5018 03/04/96 009:54 Dynamically Allocating Arrays
ti2950 asc 5230 03/04/96 009:54 Resource Expert: What It Is and How to Install It
ti2951 asc 45706 03/04/96 009:54 Delphi Configuration Files
ti2953 asc 3050 03/04/96 009:54 BDE: Writing Buffer to Disk
ti2954 asc 2335 03/04/96 009:54 Creating and Using Parameterized Queries
ti2955 asc 7536 03/04/96 009:55 Working With Auto-increment Field

Types
ti2956 asc 2526 03/04/96 009:55 How to Populate a TDBComboBox Or TDBListBox
ti2957 asc 7906 05/21/96 008:32 New Language Features in Delphi 2.0
ti2958 asc 3528 03/04/96 009:55 Preventing a Form from Resizing
ti2961 asc 2134 06/14/96 4:08p SQL: Sorting on a Calculated Column
ti2962 asc 3249 06/14/96 4:08p SQL: Using the SUBSTRING Function
ti2963 asc 1490 06/14/96 4:08p SQL: Summarizing a Calculated Column
ti2964 asc 8680 03/04/96 009:55 Managing Data Segment Size
ti2967 asc 4287 02/28/97 2:27p Validating input in TEdit components
ti2967 asc 4155 03/04/96 009:55 Validating input in TEdit components
Validating input in TEdit components
ti2970 asc 38485 05/21/96 008:32 DDE: A simple example
ti2976 asc 6138 05/21/96 008:33 TDBGrid and Multi-Selecting Records
ti2976 asc 6140 06/03/97 006:58 TDBGrid and Multi-Selecting Records
TDBGrid and Multi-Selecting Records
ti2977 asc 2847 05/21/96 008:33 Listing the field structures of a table.
ti2979 asc 2176 06/14/96 4:08p Showing deleted records in a dBASE table.
ti2980 asc 2293 06/14/96 4:08p Determining a memo's number of lines showing.
ti2981 asc 6040 05/21/96 008:33 Delphi 2.0 Install Issues
ti2988 asc 5675 06/14/96 4:08p How to Validate ISBNs
ti2989 asc 5118 06/24/96 4:04p BDE setup for Peer-To-Peer(Non-Dedicated) Networks
ti2993 asc 2570 08/05/96 2:14p Removing "Lock file has grown too large" Error
ti2996 asc 16887 05/06/96 0:53p Delphi 2.0 for Windows 95 & Windows NT Factsheet
ti3003 asc 7089 05/17/96 10:11a Delphi Client/Server Suite 2.0 for Windows 95 & NT
ti3005 asc 6241 06/14/96 4:08p Performing database queries in a background thread
ti3009 asc 1479 10/23/96 10:03a How to check a ComboBox without OnClick ocurring.
ti3050 asc 987 06/14/96 4:09p TForm.MDIChildren[] Array and Form Creation
ti3051 asc 6443 06/12/96 10:10a Delphi Client/Server Certification Program
ti3052 asc 3130 06/12/96 10:10a Authorized Client/Server Education Centers
ti3053 asc 6981 06/12/96 10:10a Delphi Client/Server 2.0 Courseware
ti3054 asc 8639 06/12/96 10:10a Study Objectives for the Delphi Client/Server Exam
ti3055 asc 4356 06/12/96 10:11a Delphi Client/Server 2.0 Train-the-Trainer Class
ti3078 asc 5161 12/13/96 1:07p Redistributing Applications using the ISP
ti3089 asc 7732 08/14/96 4:08p Sharing Violation Error with Paradox Tables
ti3096 asc 1633 08/21/96 2:04p How to Create a TDBGrid Lookup Field in Delphi 2.0
ti3097 asc 2532 08/20/96 10:09a Dynamically Creating Page Controls and Tab Sheets
ti3098 asc 780 08/20/96 10:09a Navigating a Multiselected Listbox
ti3099 asc 3311 08/20/96 10:09a Making Your Delphi 2.0 Applications

"Sing"
ti3100 asc 1832 08/20/96 10:09a Obtaining the Physical Path of a Table
ti3101 asc 1871 08/21/96 4:05p Making Accelerators Work with a TPageControl
ti3102 asc 4382 08/21/96 4:05p How to Dynamically Create A Page Control
ti3103 asc 6612 08/21/96 4:05p BDE Callbacks to Provide Status on Operations
ti3104 asc 4546 08/20/96 10:09a Accessing Paradox Tables on CD or Read-Only Drive
ti3105 asc 11317 08/21/96 10:26a Synchronize a DLL to an Open Dataset
ti3106 asc 3073 08/20/96 10:10a Clean-Boot Delphi 2.0 Installation
ti3128 asc 875 11/13/96 10:11a Creating a Wallpaper Using Delphi
ti3133 asc 1374 10/04/96 10:03a Detecting Windows Shutdown
ti3136 asc 1846 11/13/96 10:13a Returning Default Cursor after Running Queries
ti3137 asc 2617 11/13/96 10:14a Dynamic creation and circularly linking forms
ti3138 asc 5011 11/13/96 10:14a Avoid using Resource Heap with Tabbed Notebooks
ti3144 asc 8513 11/13/96 10:15a The DocOutput Object: Properties and Methods
ti3145 asc 6996 11/13/96 10:15a The DocOutput Object: Properties and Methods
The DocInput Object: Properties and Methods
ti3150 asc 2730 10/21/96 1:02p Creating Class Properties
ti3151 asc 2476 11/13/96 10:16a Optimizing Oracle Connections with Windows 95
ti3152 asc 4872 11/13/96 10:17a Connecting to a 32-bit Sybase server
ti3153 asc 6305 11/13/96 10:17a Hints on Overcoming Installation Problems
ti3155 asc 7636 11/13/96 008:10 A Better Way To Print a Form
ti3156 asc 8424 11/13/96 008:11 Creating a DataAware Control for Browsing Data
ti3157 asc 1252 11/13/96 008:11 Using InputBox, InputQuery, and ShowMessage
ti3158 asc 4473 11/13/96 008:11 Create a new file with the .wav extension.
ti3159 asc 1774 11/13/96 1:09p Using MS Internet Explorer 3.0 in Delphi 2
ti3160 asc 10624 11/20/96 11:10a BDE and Database Desktop Locking Protocol
ti3162 asc 2218 11/22/96 5:09p Getting a record member char array into a memo.
ti3164 asc 10086 12/09/96 2:08p How to Get the Most Out of DBDEMOS
ti3165 asc 2160 12/09/96 2:09p Exposing a multi string object in COM
ti3166 asc 4179 12/09/96 2:09p Getting runtime properties at runtime
ti3170 asc 2135 12/20/96 4:09p Search and replace in strings: a task made easy
ti3171 asc 6215 04/10/97 10:03a Dynamically creating a TTable & fields at runtime
ti3171 asc 6215 04/10/97 11:03a Dynamically creating a TTable & fields at runtime
ti3171 asc 6215 04/15/97 4:25p Dynamically creating a TTable & fields at runtime
ti3171 asc 6216 12/20/96 4:09p Dynamically creating a TTableand fields at runtime

Dynamically creating a TTable & fields at runtime
ti3172 asc 2262 01/22/97 11:08a Activation and Use of the CPUWindow in the IDE
ti3172 asc 2544 06/02/97 007:00 Activation and Use of the CPUWindow in the IDE
Activation and Use of the CPUWindow in the IDE
ti3187 asc 3244 01/08/97 3:59p Passing Multidimensional Arrays as Parameters
ti3188 asc 2040 01/14/97 11:08a Steps for FAT32 Support with the BDE
ti3188 asc 1927 03/28/97 11:04a Steps for FAT32 Support with the BDE
ti3188 asc 2311 04/02/97 4:03p Steps for FAT32 Support with the BDE
ti3188 asc 3043 04/03/97 008:03 Steps for FAT32 Support with the BDE
Steps for FAT32 Support with the BDE
ti3192 asc 2478 01/31/97 008:06 This document implements Drag and Drop
ti3192 asc 2479 02/03/97 10:07a Implementing Drag and Drop Functionality
ti3192 asc 2480 02/04/97 008:06 Implementing Drag and Drop Functionality
Implementing Drag and Drop Functionality
ti3194 asc 5209 02/28/97 2:27p Trapping Windows Messages in Delphi
ti3195 asc 1499 07/25/97 007:21 test document please check out & delete after appr
Hmm. a funny name, but I left the it here anyway.:-}
ti3196 asc 4385 02/28/97 2:27p Direct Commands to Printer - Passthrough/Escape
ti3197 asc 3210 02/28/97 2:27p Creating a form based on a string
ti3198 asc 1189 02/28/97 2:27p Finding the color depth of a canvas
ti3198 asc 1178 03/03/97 007:28 Finding the color depth of a canvas
Finding the color depth of a canvas
ti3199 asc 1390 02/28/97 2:27p Using FindFirst and the WIN_32_FIND_DATA structure
ti3200 asc 1572 02/28/97 2:27p Setting the pixels per inch property of TPrinter
ti3201 asc 7577 02/28/97 2:27p How to use a string table resource
ti3201 asc 7577 03/03/97 008:28 How to use a string table resource
ti3201 asc 7577 03/03/97 10:07a How to use a string table resource
How to use a string table resource
ti3202 asc 3369 03/06/97 10:08a Borland Assist for Delphi/400
ti3202 asc 3333 03/07/97 10:31a Borland Assist for Delphi/400
ti3202 asc 3333 03/10/97 007:05 Borland Assist for Delphi/400
Borland Assist for Delphi/400
ti3204 asc 2728 04/02/97 4:04p TRichEdit Printing in Delphi 2 & Windows NT 4.0
ti3209 asc 5874 04/02/97 4:05p How to use a user defined resource.
ti3209 asc 5723 04/03/97 11:04a How to use a user defined resource.
ti3209 asc 5723 04/03/97 11:25a How to use a user defined resource.
How to use a user defined resource.
ti3210 asc 3055 04/02/97 4:05p A better way to do pointer arithmetic
ti3210 asc 2924 04/03/97 11:05a A better way to do pointer arithmetic
ti3210 asc 2924 04/03/97 11:26a A better way to do pointer arithmetic
A better way to do pointer arithmetic
ti3211 asc 1566 04/02/97 4:05p Assuring Proper Font Scaling When Printing
ti3211 asc 1566 04/09/97 008:28 Assuring Proper Font Scaling When Printing
Assuring Proper Font Scaling When Printing
ti3211 asc 1566 04/09/97 10:04a Assuring Proper Font Scaling When

Printing
ti3212 asc 27912 04/01/97 11:26a BDE Error listing
ti3212 asc 27912 04/09/97 007:25 BDE Error listing
ti3212 asc 27912 04/14/97 1:25p BDE Error listing
ti3212 asc 27912 06/19/97 10:00a BDE Error listing
BDE Error listing
ti3213 asc 7171 07/07/97 007:59 Delphi 3 file types with descriptions
ti3213 asc 7171 07/07/97 008:22 Delphi 3 file types with descriptions
ti3213 asc 7171 07/07/97 009:59 Delphi 3 file types with descriptions
Delphi 3 file types with descriptions
ti3214 asc 2678 07/07/97 007:59 Moving Projects Between Machines or Directories
ti3214 asc 2678 07/07/97 008:22 Moving Projects Between Machines or Directories
ti3214 asc 2678 07/07/97 10:00a Moving Projects Between Machines or Directories
Moving Projects Between Machines or Directories
ti3215 asc 6648 07/07/97 008:00 An example of drag and drop between DBGrids
ti3215 asc 6648 07/07/97 008:22 An example of drag and drop between DBGrids
ti3215 asc 6648 07/07/97 10:00a An example of drag and drop between DBGrids
An example of drag and drop between DBGrids
ti3216 asc 1812 07/07/97 008:01 Looping Through the Controls and Components Arrays
ti3216 asc 1812 07/07/97 008:22 Looping Through the Controls and Components Arrays
ti3216 asc 1812 07/07/97 10:01a Looping Through the Controls and Components Arrays
Looping Through the Controls and Components Arrays
ti3217 asc 3492 07/07/97 008:01 Adding ODBC Drivers in Delphi 3.0
ti3217 asc 3492 07/07/97 008:22 Adding ODBC Drivers in Delphi 3.0
ti3217 asc 3492 07/07/97 10:01a Adding ODBC Drivers in Delphi 3.0
Adding ODBC Drivers in Delphi 3.0
ti3218 asc 1623 07/07/97 008:01 Delphi/400: Activating your License Key
ti3218 asc 1623 07/07/97 008:22 Delphi/400: Activating your License Key
ti3218 asc 1623 07/07/97 10:02a Delphi/400: Activating your License Key
Delphi/400: Activating your License Key
ti3231 asc 3911 07/22/97 3:51p Displaying System Resources in Win 95 and NT 4
ti3232 asc 2765 07/22/97 3:52p Edit Controls that Align Under NT 4
ti3233 asc 1708 07/22/97 3:52p How do I map a network drive in Windows NT or '95?
ti3234 asc 2808 07/22/97 3:52p Adding shortcuts to Win95/WinNT4 Desktop/StartMenu
ti3235 asc 2111 07/22/97 3:52p Minimizing Application When a Form Minimizes
ti3239 asc 3211 08/11/97 009:54 Graying Out Enabled/Disabled Data Aware Controls
ti3240 asc 9202 08/11/97 009:54 Exposing a multi string object in COM
ti3241 asc 1781 08/11/97 009:54 Getting Version Information From Your Program
ti8520 asc 4191 08/21/96 10:27a Delphi Titles from Leading Book

Publishers
ti9604 asc 3034 04/25/96 10:19a An Overview of Borland Online Information Services
ti9605 asc 7073 01/13/97 2:16p Technical Support & Customer Support Phone Numbers
ti9605 asc 7487 02/04/97 2:08p Technical Support & Customer Support Phone Numbers
ti9605 asc 7603 03/05/97 10:08a Technical Support & Customer Support Phone Numbers
ti9605 asc 7543 03/21/97 4:06p Developer Support & Customer Support Phone Numbers
ti9605 asc 7543 03/24/97 006:07 Developer Support & Customer Support Phone Numbers
ti9605 asc 3924 06/05/97 10:02a Developer Support & Customer Support Phone Numbers
Developer Support & Customer Support Phone Numbers
ti9652 asc 8614 04/23/96 3:53p Borland International's TechFax System
ti9656 asc 9312 04/25/96 10:21a Technical Support Via the Internet
ti9677 asc 1764 04/25/96 10:22a Borland's Technical Suport Download Bulletin Board
ti9800 asc 14748 01/13/97 4:08p Borland Assist
ti9800 asc 14915 02/04/97 2:12p Borland Assist
ti9800 asc 15042 03/05/97 10:12a Borland Assist
ti9800 asc 15153 03/06/97 2:28p Borland Assist
ti9800 asc 15116 03/07/97 10:31a Borland Assist
ti9800 asc 15116 03/10/97 007:05 Borland Assist
ti9800 asc 24569 05/13/97 4:02p Borland Assist
ti9800 asc 24684 06/05/97 10:04a Borland Assist
Borland Assist

About Delphi Tis

Welcome to Delphi TIs, a list of the Borland Delphi Technical Information Documents, brought to you by **D C AL CODA**.

Contained in this Help file are all the Delphi TI files that were found on the Borland FTP site, as of the Help File creation date. If there is enough interest and we have the time, this Help file will be periodically updated.

This Help File is provided as is, with no guarantee that the TIs contained are accurate. You use this information and Help file at your own risk. We will not be help liable for any harm you think use of this information or Help file has or has not caused.

We have left the TIs just as they were downloaded from Borland's FTP site. The _Index file was edited, to remove references to the zipped files. There seems to be some duplication of a few TIs, maybe due to revised TIs being published. The _Index file is really of limited use in this Help file, but it has been included anyway.

Using this Help file:

Of course you all know how to use a help file.:-}

But, one suggestion is to make a Find word list (database) in Windows 95. You can then do a Find on any words and easily view the results.

History:

9/21/1997:
Initial release to a few Delphi Mailing Lists

9/28/1997:
Made the Contents topic the Index of all included TIs
Added links to the Index of TIs, so you can browse the Index and jump to the TI from there

9/29/1997:
Compiled it without compression, because on my system, I could not create a Find database with the Help File compiled in compressed form.

Errors:

If you find errors in the actual working or layout of the Help File, please let us know at:

kenhale@dcalcoda.com

We hope to keep the latest version, if we can afford the space, on the **D C AL CODA** web site:

http://www.dcalcoda.com/

We will make this file available on some of the great Delphi Web sites.

If you find errors in the actual TI text or code, I am not sure what you should do.:-}

Thanks:

This Help file is given to the international Delphi community in appreciation for all the help we have gotten and that happens on a day to day basis. It is a joy to program in Delphi and a joy to see such help and communication going on.

Thanks also to Jan Goyvaerts for writing HelpScribble (in Delphi!), which was to create this Help file.

Thank you!

Ken Hale and Coda Hale
kenhale@dcalcoda.com
http://www.dcalcoda.com/

A VERY short primer on dynamic memory allocation.

NUMBER : 2647
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : A VERY short primer on dynamic memory allocation.

Q: How do I reduce the amount of memory taken from the data segment? (or How do I allocate memory dynamically?)

A:

Let's say your data structure looks like this:

```
type
  TMyStructure = record
    Name: String[40];
    Data: array[0..4095] of Integer;
  end;
```

That's too large to be allocated globally, so instead of declaring a global variable,

```
var
  MyData: TMyStructure;
```

you declare a pointer type,

```
type
  PMyStructure = ^TMyStructure;
```

and a variable of that type,

```
var
  MyDataPtr: PMyStructure;
```

Such a pointer consumes only four bytes of the data segment.

Before you can use the data structure, you have to allocate it on the heap:

```
New(MyDataPtr);
```

and now you can access it just like you would global data. The only difference is that you have to use the caret operator to dereference the pointer:

```
MyDataPtr^.Name := 'Lloyd Linklater';
MyDataPtr^.Data[0] := 12345;
```

Finally, after you're done using the memory, you deallocate it:

```
Dispose(MyDataPtr);
```

Installing Delphi on a Network

NUMBER : 2558
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Installing Delphi on a Network

----Installing Delphi to a network drive-----

Note:

Delphi was never intended to be run from a network, and Borland makes no claims in it's ability to do so. This information is provided solely as a starting point and is NOT intended to be a cookbook.

This TI is assuming a basic Windows 3.1 or 3.11 installation. if you are using another operating system, appropriate adjustments will, of course, need to be made.

****

1. Install Delphi to the Network from any Workstation, but be sure to check the "Windows Lan" option if Windows is installed on a server and it's directories are read only.

2. Edit DELPHI.INI in the windows directory.
   -enter:
   ```
   [Globals]
   PrivateDir = your local directory
   ```
   -add:
   ```
   [Library]
   SearchPath = Netdrive:\DELPHI\LIB
   ComponentLibrary= your local directory\COMPLIB.DCL
   ```

   This will allow each user to have his own private options, private DFM file, as well as a private Component Library. If a public component library is desired, the ComponentLibray line may be left out.

3. Copy COMPLIB.DCL and DELPHI.DMT from the DELPHI\BIN directory to the local private directories.

4. Copy DELPHI.INI to all machines to be running Delphi.

   If Windows is on the network, everyone will need to set up their local directories identically, since it will only have that one INI file to work from.

5. If the WINSYS option was selected at install time, there was a DELPHI\WINSYS directory created. There is a bug in the install engine, and not all of the needed files get included. Copy the files from: CDROM:\RUNIMAGE\WINDOWS and

CDROM:\RUNIMAGE\WINDOWS\SYSTEM into the WINSYS directory. Then this directory will need to be included in the path. If you wish to copy these files into the WINDOWS\SYSTEM directory, be sure to back up those directories first, in case of file collisions. Care should be taken to not overwrite newer versions of these files. You may find the DOS program REPLACE.EXE useful in accomplishing this.

6. If there was no WINSYS directory created (Step 5), Copy CD:RUNIMAGE\WINDOWS and CD:\RUNIMAGE\WINODWS\SYSTEM directories to all Workstations. Put these files in a separate directory and include it in the path.(This will make uninstalling much easier.)

7. Copy the group file created in the windows directory to all other stations to be using Delphi.

8. Use Program Manager to create the group on these machines.

9. Modify the path on all workstations that will run Delphi as follows:

   PATH=C:\IBLOCAL\BIN;C:\DELPHI\BIN

   All users must load the DOS command SHARE(Assuming windows 3.1) before using Delphi.

10. The following line should be added to each user's AUTOEXEC.BAT:

    SHARE /F:4096 /L:40

11. Modify the WIN.INI on all machines (assuming a local windows installation,) adding the following:

    [IDAPI]
    DLLPATH=X:\IDAPI;C:\IDAPI
    CONFIGFILE01=X:\IDAPI\IDAPI.CFG

    [Borland Language Drivers]
    LDPath=X\IDAPI\LANGDRV

    [BWCC]
    BitmapLibrary=BWCC.DLL

    [Interbase]
    RootDirectory=X:\IBLOCAL

    [Paradox Engine]
    UserName=PxEngine
    NetNamePath=X:\
    MaxTables=64
    RecBufs=64
    MaxLocks=64
    MaxFiles=64
    SwapSize=64
    NetNameFDSM=

```
[DDE Servers]
DBD=X:\DBD\DBD

[DBD]
WORKDIR=X:\DBD
PRIVDIR=C:\DBD\DBDPRIV
```

Where X is the Network Drive and path installed to.
Note that PRIVDIR is set locally.

Note, Delphi will install numerous INI files to the Windows directory. These files can be copied over to other workstations only if the corresponding INI files do not already reside there. (They shouldn't) If this is the case you will have to manually attach these files to the ends of the existing ones.

The INI files that Delphi installs are:

RS_SQLIF INI
WINHELP INI
MULTIHLP INI
DELPHI INI
ODBCINST INI
ODBC INI
RPTSMITH INI
RS_RUN INI
ODBCISAM INI

The configuration described above should allow for the installed location to be set read only after it is installed, but this was only tested with a local installation of Windows.

Making a form selectable without the main form.

NUMBER : 2602
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Making a form selectable without the main form.

Q: How do I make it so that only the form I select comes to the top? (i.e. without the main form)

A: Try this in any secondary window that you DON'T want dragging the program along:

```
...
private {This goes in the for's type declaration.}
  { Private declarations }
  procedure CreateParams(VAR Params: TCreateParams); override;
...

procedure TForm2.CreateParams(VAR Params: TCreateParams);
begin
  Inherited CreateParams(Params);
  Params.WndParent := GetDesktopWindow;
end;
```

By setting the form's parent window handle to the desktop, you remove the link that would normally force the whole application to come to the top when this form comes to the top.

How to get a file's date and time stamp

NUMBER : 2616
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to get a file's date and time stamp.

Q: How do I get a file's date and time stamp?

A:

```
function GetFileDate(TheFileName: string): string;
var
  FHandle: integer;
begin
  FHandle := FileOpen(TheFileName, 0);
  try
    Result := DateTimeToStr(FileDateToDateTime(FileGetDate(FHandle)));
  finally
    FileClose(FHandle);
  end;
end;
```

Lists the virtual key values

NUMBER : 2617
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Lists the virtual key values

Q: What are the values for the virtual keys?

A:

```
  vk_LButton    = $01;
  vk_RButton    = $02;
  vk_Cancel     = $03;
  vk_MButton    = $04;    { NOT contiguous with L & RBUTTON }
  vk_Back       = $08;
  vk_Tab        = $09;
  vk_Clear      = $0C;
  vk_Return     = $0D;
  vk_Shift      = $10;
  vk_Control    = $11;
  vk_Menu       = $12;
  vk_Pause      = $13;
  vk_Capital    = $14;
  vk_Escape     = $1B;
  vk_Space      = $20;
  vk_Prior      = $21;
  vk_Next       = $22;

  vk_End        = $23;
  vk_Home       = $24;
  vk_Left       = $25;
  vk_Up         = $26;
  vk_Right      = $27;
  vk_Down       = $28;
  vk_Select     = $29;
  vk_Print      = $2A;
  vk_Execute    = $2B;
  vk_SnapShot   = $2C;
{ vk_Copy       = $2C not used by keyboards }
  vk_Insert     = $2D;
  vk_Delete     = $2E;
  vk_Help       = $2F;
{ vk_A thru vk_Z are the same as their ASCII equivalents: 'A' thru 'Z' }
{ vk_0 thru vk_9 are the same as their ASCII equivalents: '0' thru '9' }

  vk_NumPad0    = $60;
  vk_NumPad1    = $61;
  vk_NumPad2    = $62;
  vk_NumPad3    = $63;
  vk_NumPad4    = $64;
  vk_NumPad5    = $65;
```

```
  vk_NumPad6     = $66;
  vk_NumPad7     = $67;
  vk_NumPad8     = $68;
  vk_NumPad9     = $69;
  vk_Multiply    = $6A;
  vk_Add          = $6B;
  vk_Separator = $6C;
  vk_Subtract    = $6D;
  vk_Decimal     = $6E;
  vk_Divide      = $6F;
  vk_F1           = $70;
  vk_F2           = $71;
  vk_F3           = $72;
  vk_F4           = $73;
  vk_F5           = $74;

  vk_F6           = $75;
  vk_F7           = $76;
  vk_F8           = $77;
  vk_F9           = $78;
  vk_F10          = $79;
  vk_F11          = $7A;
  vk_F12          = $7B;
  vk_F13          = $7C;
  vk_F14          = $7D;
  vk_F15          = $7E;
  vk_F16          = $7F;
  vk_F17          = $80;
  vk_F18          = $81;
  vk_F19          = $82;
  vk_F20          = $83;
  vk_F21          = $84;
  vk_F22          = $85;
  vk_F23          = $86;
  vk_F24          = $87;
  vk_NumLock     = $90;
  vk_Scroll      = $91;
```

How to paint the form with a bitmap.

NUMBER : 2619
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to paint the form with a bitmap.

Q: How do I paint the background of my form with a bitmap as tiles?

A:

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  Bitmap: TBitmap;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('C:\WINDOWS\cars.BMP');
end;

procedure TForm1.FormPaint(Sender: TObject);
var
  X, Y, W, H: LongInt;
begin
  with Bitmap do begin
    W := Width;
    H := Height;
  end;
  Y := 0;
```

```
  while Y < Height do begin
    X := 0;
    while X < Width do begin
      Canvas.Draw(X, Y, Bitmap);
      Inc(X, W);
    end;
    Inc(Y, H);
  end;
end;

end.
```

How to associate a string with a component.

NUMBER : 2621
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to associate a string with a component.

Q: Is there a way to associate a string with each component?

A: Since the Tag property is a longint, you can type cast it as a Pointer or PChar. So, you can basically store a pointer to a record by using the Tag property.

Note: You're not going to be able to store the string, or pointer rather, at design time. This is something you'll have to do at run time. Take a look at this example:

```
var
  i: integer;
begin
   for i := 0 to ComponentCount - 1 do

      if Components[i] is TEdit then
         Components[i].Tag := LongInt(NewStr('Hello '+IntToStr(i)));
end;
```

Here, I loop through the components on the form. If the component is a TEdit, I assign a pointer to a string to its Tag property. The NewStr function returns a PString (pointer to a string). A pointer is basically the same as a longint or better, occupies the same number of bytes in memory. Therefore, you can type cast the return value of NewStr as a LongInt and store it in the Tag property of the TEdit component. Keep in mind that this could have been a pointer to an entire record. Now I'll use that value:

```
var
  i: integer;
begin
   for i := 0 to ComponentCount - 1 do
      if Components[i] is TEdit then begin
         TEdit(Components[i]).Text := PString(Components[i].Tag)^;
         DisposeStr(PString(Components[i].Tag));
      end;
end;
```

Here, again I loop through the components and work on only the TEdits. This time, I extract the value of the component's Tag property by typecasting it as a PString (Pointer to a string) and assigning that value to the TEdit's Text property. Of course, I must dereference it with the caret (^) symbol. Once I do that, I dispose of the string stored in the edit

component. Important note: if you store anything in the TEdit's Tag property as a pointer, you are responsible for disposing of it also.

FYI, Since Delphi objects are really pointers to class instances, you can also store objects in the Tag property. As long as you remember to Free them.

Three methods spring to mind to use Tags to access strings that persist from app to app.

1. If your strings stay the same forever, create a string resource in Resource Workshop (or equiv) and use the Tags as indexes into your string resource.

2. Use TIniFile and create a section for your strings, and give each string a name with number so that your ini file has a section like this:

```
[strings]
string1=Aristotle
string2=Plato
string3=Well this is Delphi, after all
```

Then you can fetch them back out this way:

```
var s1: string;
...
s1 := IniFile1.ReadString('strings', 'string'+IntToStr(Tag), '');
```

3. Put your strings into a file, with each followed by a carriage return. Read them into a TStringList. Then your Tags become an index into this stringlist:

```
StringList1.LoadFromFile('slist.txt');
...
s1 := StringList1[Tag];
```

Given the way Delphi is set up, I think the inifile method is easiest.

How close a file opened from a Delphi DLL in VB.

NUMBER : 2622
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How close a file opened from a Delphi DLL in VB.

Q: How do I close a file that was opened in a DLL (Delphi made) and called from VB?

A: This is a known problem. It comes from the fact that VB closes the 5 DOS standard handles (0..4) at startup. So the open file routine will reuse one of these handles to open the first disk file. That is not a problem in using the file, but the Pascal Close routine has a build-in safety feature: it refuses to close a file that has one of the standard handles! That is a Good Thing under DOS but screws up the works in your situation since the file opened by the DLL is never closed, not even when the DLL goes down! VC++ is obviously less restricted and will close a standard handle.

You can fix this problem yourself. Instead of using the Pascal Close/CloseFile routine to close the file in the DLL, use one of these:

```
Procedure ReallyCloseFileVar(Var F); Assembler;
{ F should be a file type }
Asm
  les   bx, F                    { store F in es:bx }
  mov   bx, word ptr es:[bx] { store handle in bx }
  mov   ah, $3E                    { function 3Eh = close file }
  call Dos3Call                  { execute int 21h }
End;

Procedure ReallyCloseFileHandle(FileHandle: word); assembler;
{ FileHandle is the DOS file handle }
asm
  mov   bx, Handle { store handle in bx }
  mov   ah, $3E      { function 3Eh = close file }
  call DOS3Call    { execute int 21h }
end;
```

How to readln longer than 255 chars.

NUMBER : 2645
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to readln longer than 255 chars.

Q: How can I readln() from a file when the lines are longer than 255 bytes?

A: ReadLn will accept an array [0..something] of Char as buffer to put the read characters in and it will make a proper zero-terminated char out of them. The only limitation is this: the compiler needs to be able to figure out the size of the buffer at compile time, which makes the use of a variable declared as PChar and allocated at run-time impossible.

Workaround:

```
Type
   {use longest line you may encounter here}
   TLine = Array [0..1024] of Char;

   PLine = ^TLine;

Var
   pBuf: PLine;
...
   New( pBuf );

...
   ReadLn( F, pBuf^ );
```

To pass pBuf to functions that take a parameter of type Pchar, use a typecast like PChar(pBuf).

Note: you can use a variable declared as of type TLine or an equivalent array of char directly, of course, but I tend to allocate anything larger than 4 bytes on the heap...

Uninstalling Delphi (16 bit)

NUMBER : 2529
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Uninstalling Delphi

Uninstalling Delphi (16 bit)

Contents

Introduction

This document describes the process of "uninstalling" Delphi from your system. It is a good idea to have a back up of your system files, due to the fact that you will need to use the 'delete' command.

This document assumes the product was installed to C: drive, with default directories and full (default) installation settings. Any deviations from the default settings may require adjustments/ changes to the instructions below.

A) Deleting Directories

If you have any program that uses IDAPI, please disregard any mention of deleting it or its directory below.

If you wish to delete directories using DOS commands, see section 1 below. To use Windows, see section 2.

1) To erase directories from your hard drive on a system with DOS 6.0 or greater, you need to be in the root directory (i.e. C:\>) and type DELTREE <DIR NAME> and hit return(Example: C:\>deltree delphi). Answer Yes when it prompts to delete all subdirectories (see below for the list of directories to delete). For versions of DOS lower than 6.0, you need to go into each directory and type DEL *.* until all directories are empty. Then DOS will allow you to delete the directory names with the command RD <DIR NAME>.

2) From Windows, go to File Manager. Using File Manager, select the <DIR NAME> folder. Then hit the Delete key. Answer Yes to All when prompted to delete subdirectories.

The directories you need to delete are:

C:\Delphi
C:\Rs_run
C:\Iblocal
* C:\Informix
C:\Rptsmith
C:\Dbd
C:\Idapi

* - Client/ Server version only

B) Deleting the Program Group

To delete the program group 'Delphi' in Windows 3.1 and Windows NT, highlight the group without opening it, and press the Delete key. Under Windows 95, the folders will have been removed by deleting the directories in the previous section.

C) Removing .INI files

Delete the following .INI files from your \WINDOWS directory (CAUTION: if you have any other Borland products or any programs that use ODBC, back up the .INI files listed below before deleting them. Then test all other programs to insure they run normally!):

DELPHI.INI
BORHELP.INI
DBD.INI
INTERBAS.INI
* ODBC.INI
* ODBCINST.INI
* ODBCISAM.INI
RPTSMITH.INI
RS_RUN.INI
RS_SQLIF.INI
VSL.INI
MULTIHLP.INI

* - files used by ODBC applications

D) Editing WIN.INI

Make a backup of your WIN.INI file. Then, in your \WINDOWS directory, remove the following sections from your WIN.INI file(CAUTION: If any other program uses Paradox Engine, IDAPI, or Database Desktop, use discretion in deleting

these lines!):

```
[DDE Servers]
DBD=C:\DBD\DBD

[DBD]
WORKDIR=C:\DBD
PRIVDIR=C:\DBD\DBDPRIV

[Interbase]
RootDirectory=C:\IBLOCAL

* [IDAPI]
DLLPATH=C:\IDAPI
CONFIGFILE01=C:\IDAPI\IDAPI.CFG

* [Borland Language Drivers]
LDPath=C:\IDAPI\LANGDRV
```

* - DO NOT delete these if other programs use IDAPI

Remember to save your new WIN.INI.

E) Editing WINHELP.INI

Remove the following lines from WINHELP.INI located in your WINDOWS directory:

```
BDECFG.HLP=C:\IDAPI
DBD.HLP=C:\DBD
DELPHI.HLP=C:\DELPHI\BIN
WINAPI.HLP=C:\DELPHI\BIN
CWG.HLP=C:\DELPHI\BIN
CWH.HLP=C:\DELPHI\BIN
LOCALSQL.HLP=C:\DELPHI\BIN
VQB.HLP=C:\DELPHI\BIN
SQLREF.HLP=C:\IBLOCAL\BIN
WISQL.HLP=C:\IBLOCAL\BIN
RPTSMITH.HLP=C:\rptsmith
RS_DD.HLP=C:\rptsmith
SBL.HLP=C:\rptsmith
RS_RUN.HLP=C:\rptsmith
RCEXPERT.HLP=C:\DELPHI\RCEXPERT
DRVDBASE.HLP=C:\WINDOWS\SYSTEM
DRVPARDX.HLP=C:\WINDOWS\SYSTEM
ODBCINST.HLP=C:\WINDOWS\SYSTEM
```

F) Editing your AUTOEXEC.BAT

First, make a backup of your AUTOEXEC.BAT file. Then edit your AUTOEXEC.BAT and remove C:\IBLOCAL\BIN; and C:\IDAPI; from

your PATH statement if they are present.
(Example:
  (OLD) PATH = C:\;C:\WINDOWS;C:\DOS;C:\IDAPI;C:\IBLOCAL\BIN;
  (NEW) PATH = C:\;C:\WINDOWS;C:\DOS;)
Remember to save your new AUTOEXEC.BAT.

G) Removing files from your WINDOWS and WINDOWS\SYSTEM directories

Please note: These files are listed for reference ONLY. It IS NOT recommended that you delete these files. Deletion of certain files used by other programs WILL cause your system to crash.

CAUTION!!!!CAUTION!!!!CAUTION!!!!CAUTION!!!!CAUTION!!!!CAUTION!!!!

Many files shown in this section are shared by other programs. Deleting them may cause other programs not to work INCLUDING WINDOWS. If you wish to remove files from these directories, please do so ONLY at your own discretion. One possible method would be to delete a file, and run ALL other programs on your system to make sure they work. Copies of all files listed below are located on the Delphi CD in the runimage directory. You may copy files from the CD if one that is required is inadvertantly deleted. Please be careful.

CAUTION!!!!CAUTION!!!!CAUTION!!!!CAUTION!!!!CAUTION!!!!CAUTION!!!!

Files placed or updated in your \WINDOWS directory by Delphi:

DELPHI.CBT	M3OPEN.DLL
M3OPEN.EXE	MBW.EXE
MFTP.EXE	MHPARPA.DLL
MNETONE.EXE	MNOVLWP.DLL
MPATHWAY.DLL	MPCNFS.EXE
MPCNFS2.EXE	MPCNFS4.DLL
MSOCKLIB.DLL	MVWASYNC.EXE
MWINTCP.EXE	WINSOCK.DLL

Files placed or updated in your \WINDOWS\SYSTEM directory by Delphi:

BIBAS04.DLL	BIFLT04.DLL
BIMDS04.DLL	BIPDX04.DLL
BIPDX04.HLP	BIUTL04.DLL
BIVBX11.DLL	BLBAS04.DLL
BLINT04.DLL	BLINT04.HLP
BLMDS04.DLL	BLODBC.LIC
BLUTL04.DLL	BOLE16D.DLL
BTRV110.DLL	BWCC.DLL
CHART2FX.VBX	COMPOBJ.DLL
* COREWIN.DLL	CTL3D.DLL
CTL3DV2.DLL	* DBNMP3.DLL
DRVACCSS.HLP	DRVBTRV.HLP
DRVDBASE.HLP	DRVEXCEL.HLP
DRVFOX.HLP	DRVTEXT.HLP

GAUGE.VBX	* LDLLSQLW.DLL
MSJETDSP.DLL	MULTIHLP.DLL
NWCALLS.DLL	NWIPXSPX.DLL
* NWNETAPI.DLL	ODBC.DLL
ODBCADM.EXE	ODBCCURS.DLL
ODBCINST.DLL	ODBCINST.HLP
OLE2.DLL	OLE2.REG
OLE2CONV.DLL	OLE2DISP.DLL
OLE2NLS.DLL	OLE2PROX.DLL
* ORA6WIN.DLL	* ORA7WIN.DLL
PICT.VBX	PXENGCFG.EXE
PXENGWIN.DLL	QEBI.LIC
RED110.DLL	SIMADMIN.DLL
SIMBA.DLL	STDOLE.TLB
STORAGE.DLL	STRESS.DLL
SWITCH.VBX	TXTISAM.DLL
TYPELIB.DLL	* W3DBLIB.DLL
XBS110.DLL	XLSISAM.DLL

* - Client/ Server version only

All .BOR files are backup files that Delphi renamed and replaced with updated or modified files, you may use these files in comparison to the files you have changed. If you have installed any software after the .BOR file was created or the .BOR file is the same as the file it replaced, the .BOR file can be deleted.

How to use array of const.

NUMBER : 2654
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to use array of const.

Q: How do I use "array of const"?

A: An array of const is in fact an open array of TVarRec (a predeclared Delphi type you can look up in the online help). So the following is Object Pascal psuedocode for the general battle plan:

```
procedure AddStuff( Const A: Array of Const );
Var i: Integer;
Begin
  For i:= Low(A) to High(A) Do
  With A[i] Do
    Case VType of
    vtExtended: Begin
        { add real number, all real formats are converted to
          extended automatically }
      End;
    vtInteger: Begin

        { add integer number, all integer formats are converted
          to LongInt automatically }
      End;
    vtObject: Begin
        If VObject Is DArray Then
          With DArray( VObject ) Do Begin
            { add array of doubles }
          End
        Else If VObject Is IArray Then
          With IArray( VObject ) Do Begin
            { add array of integers }
          End;
      End;
    End; { Case }
End; { AddStuff }
```

For further information see "open arrays" in the on-line help.

Function mapping from the Paradox Engine to BDE

NUMBER : 2656
PRODUCT : BDE
VERSION : 2.x
OS : Windows
DATE : August 5, 1996

TITLE : Function mapping from the Paradox Engine to BDE

Types

Paradox Engine	BDE	Variable name in this doc
PXCODE	DBIResult	rslt
TABLEHANDLE	hDBICur	tabH
RECORDHANDLE	pBYTE	recH
FIELDNUMBER	UINT16	iFld
LOCKHANDLE	UINT32	
RECORDNUMBER	UINT32	recNum
BLOBHANDLE	INT16	

Functions

Paradox Engine	BDE
BLANK	DbiPutField(tabH, iFld, recH, NULL); (Cannot compare Field Data with BLANK in IDAPI. Use the value returned from DbiGetField in bIsBlank to determine if a field is blank.)
ISBANK()	Use the blank parameter in DbiGetField(tabH, iFld, recH, pDest, &bIsBlank); or DbiVerifyField(tabH, iFld, pSrc, &bIsBlank);
PXBlobClone()	Not Supported. (Private BLOBs are not supported in IDAPI)
PXBlobClose() accept == FALSE accept == TRUE	 DbiFreeBlob(tabH, recH, iFld); Not required in IDAPI. Note that DbiFreeBlob(tabH, recH, iFld) needs to be called _after_ adding a record containing a BLOB to a table.
PXBlobDrop()	DbiGetRecord(tabH, dbiNOLOCK, recH, NULL); DbiTruncateBlob(tabH, recH, iFld, 0); DbiModifyRecord(tabH, recH, FALSE);
PXBlobGet()	DbiGetBlob(tabH, recH, iFld, iOff, iLen,

```
                                    pDest, &iRead);

PXBlobGetSize()         DbiGetBlobSize(tabH, recH, iFld, piSize);

PXBlobOpenRead()        DbiOpenBlob(tabH, recH, iField,dbiREADONLY);

PXBlobOpenWrite()       DbiOpenBlob(tabH, recH, iField,
                                    dbiREADWRITE);

PXBlobPut()             DbiPutBlob(tabH, recH, iFld, iOff, iLen,
                                    pSrc);

PXBlobQuickGet()        DbiGetBlobHeading(tabH, iFld, recH, &iSize);
                        (Paradox Tables only.)

PXDateDecode()          DbiDateDecode(dateD, piMon, piDay, piYear);

PXDateEncode()          DbiDateEncode(iMon, iDay, iYear, pdateD);

PXErrMsg()              DbiGetErrorString(iErrorCode, szError);
                        or
                        DbiGetErrorInfo(&ErrInfo);
                        (Must be called imediately after the
                         offending function - privides more
                         information than DbiGetErrorString.)

PXExit()                DbiCloseDatabase(&hDb);
                        DbiExit();

PXFldBlank()            DbiGetField(tabH, iFld, recH, NULL,
                                    &bIsBlank);
                        or
                        DbiVerifyField(tabH, iFld, pSrc, &bIsBlank);
                        bIsBlank = TRUE is field is blank.

PXFldHandle()           DbiGetFieldDescs(tabH, &fldDesc);
                        fldDesc.iFldNum;

PXFldName()             DbiGetFieldDescs(tabH, &fldDesc);
                        fldDesc.szName;

PXFldType()             DbiGetFieldDescs(tabH, &fldDesc);
                        fldDesc.iFldType;
                        fldDesc.iSubType;

PXGetAlpha()            DbiGetField(tabH, iFld, recH,
                                    (pBYTE)szString, &bIsBlank);

PXGetDate()             DbiGetField(tabH, iFld, recH,
                                    (pBYTE)&Date, &bIsBlank);

PXGetDefaults()         DbiOpenCfgInfoList(hCfg, eOpenMode,
                                           eConfigMode, pszCfgPath,
                                           &tabH);

PXGetDoub()             DbiGetField(tabH, iFld, recH,
```

(pBYTE)&Double, &bIsBlank);

PXGetLong() DbiGetField(tabH, iFld, recH,
(pBYTE)&Long, &bIsBlank);

PXGetShort() DbiGetField(tabH, iFld, recH,
(pBYTE)&Short, &bIsBlank);

PXInit() Not Supported. See PXWinInit.

PXKeyAdd() DbiAddIndex(hDb, tabH, szTblName, szTblType,
&IdxDesc);

PXKeyDrop() DbiDeleteIndex(hDb, tabH, szTblName,
szTblType, NULL, NULL,
iIndexId);

PXKeyMap() Not Supported.
(Key Mapping is no longer needed - all
pertinent information is put in the index
descriptor - IdxDesc)

PXKeyNFlds() DbiGetIndexSeqNo(tabH, NULL, NULL, 0,
&seqNo);
DbiGetIndexDesc(tabH, seqNo, &idxDesc);
idxDesc.iFldsInKey;
(Note that Paradox in the only database
which has the concept of a primary index)

PXKeyQuery() Not Supported.
(The BDE does not have any function like
this. You need to call DbiGetIndexDescs()
and then look for the index you want. Once
found all the information about that index
is available to you in the structure.)

PXNetErrUser() DbiErrGetErrorContext(ecUSERNAME,
szUserName);

PXNetFileLock() DbiAcqPersistTableLock(hDb, (pCHAR)fileName,
Driver);

PXNetFileUnlock() DbiRelPersistTableLock(hDb, (pCHAR)fileName,
Driver);

PXNetInit() Not Support. See PXWinInit.

PXNetRecGotoLock() Not Supported.
(Can be simulated by setting a bookmark on
the record which is locked and then
switching to that bookmark:
DbiSetToBookMark(tabH, pBookMark);)

PXNetRecLock() DbiGetRecord(tabH, dbiWRITELOCK, NULL,
NULL);

PXNetRecLocked() DbiIsRecordLocked(tabH, edbiLock, piLocks);

PXNetRecUnlock() DbiRelRecordLock(tabH, FALSE);
or durring the update of the record -
DbiModifyRecord(tabH, recH, TRUE);

PXNetTblChanged() Need to register a cbTABLECHANGED callback.
(Paradox only.)

PXNetTblLock() DbiAcqTableLock(tabH, eLockType);
(Note that only Read and Write locks
are supported by this function. For a
FL on a table, open the table with
the dbiOPENEXCL parameter or use the
DbiAcqPersistTableLock function.)

PXNetTblRefresh() DbiForceReread(tabH);

PXNetTableUnlock() DbiRelTableLock(tabH, eLockType);

PXNetUserName() DbiGetNetUserName(pzName);

PXPswAdd() DbiAddPassword(szPassword);
(Paradox tables only.)

PXPswDel() DbiDropPassword(szPassword);
(Paradox tables only.)

PXPutAlpha() DbiPutField(tabH, iFld, recH,
(pBYTE)szString);

PXPutBlank() DbiPutField(tabH, iFld, recH, NULL);

PXPutDate() DbiPutField(tabH, iFld, recH,
(pBYTE)&Date);

PXPutDoub() DbiPutField(tabH, iFld, recH,
(pBYTE)&Double);

PXPutLong() DbiPutField(tabH, iFld, recH,
(BYTE)&Long);

PXPutShort() DbiPutField(tabH, iFld, recH,
(pBYTE)&Short);

PXRawGet() DbiSetProp((hDBIObj)tabH, curXLTMODE,
xltNONE)
DbiGetRecord(tabH, NULL, recH, NULL);

PXRawPut() DbiSetProp((hDBIObj)tabH, curXLTMODE,
xltNONE)
DbiInsertRecord(tabH, dbiNOLOCK, recH,
pDest);

PXRecAppend() DbiAppendRecord(tabH, recH);

PXRecBufClose() The application needs to release the memory associated with the record buffer. In 'C', call: free(recH). (Record buffers in IDAPI are owned by the application.)

PXRecBufCopy() The application needs to copy the memory which is used for the record buffer. In 'C', call: memcpy(recHDest, recHSource, size). (Record buffers in IDAPI are owned by the application.)

PXRecBufEmpty() DbiInitRecord(tabH, recH);

PXRecBufOpen() The application needs to allocate memory for record buffer. In 'C', call:

```
DbiGetCursorProps(tabH, &CurProps);
recH = (pBYTE)malloc(CurProps.iRecBufSize
                     * sizeof(BYTE));
```

(Record buffers in IDAPI are owned by the application.)

PXRecDelete() DbiDeleteRecord(tabH, NULL);

PXRecFirst()

```
DbiSetToBegin(tabH);
DbiGetNextRecord(tabH, dbiNOLOCK, NULL,
                 NULL);
```

(IDAPI has the concept of Beginning of file, which is before the first record).

PXRecGet() DbiGetRecord(tabH, dbiNOLOCK, recH, NULL);

PXRecGoto() Not supported by the SQL Drivers. Example for use with 'C':

```
DBIResult gotoRec(hDBICur tabH,
                  UINT32 recNum)
{
    DBIResult    lastError;
    CURProps     curProps;

    lastError = DbiGetCursorProps(tabH,
                                  &curProps);
    if (lastError != DBIERR_NONE)
    {
        return lastError;
    }

    if (!strcmp(curProps.szTableType,
                szPARADOX))
    {
        lastError = DbiSetToSeqNo(tabH,
                                  recNum);
    }
    else if (!strcmp(curProps.szTableType,
                     szDBASE))
```

```
                    {
                        lastError = DbiSetToRecordNo(tabH,
                                                     recNum);
                    }
                    else
                    {
                        lastError = DBIERR_NOTSUPPORTED;
                    }

                    return lastError;
                }
```

PXRecInsert()

```
DbiInsertRecord(tabH, dbiNOLOCK, recH);
```

PXRecLast()

```
DbiSetToEnd(tabH);
DbiGetPriorRecord(tabH, dbiNOLOCK, NULL,
                  NULL);
```

(IDAPI has the concept of End of File, which is after the last record).

PXRecNext()

```
DbiGetNextRecord(tabH, dbiNOLOCK, NULL,
                 NULL);
```

PXRecNFlds()

```
DbiGetCursorProps(tabH, &CurProps);
lFlds = CurProps.iFields;
```

PXRecNum()

Not supported for SQL Tables.
Example for use with 'C':

```
DBIResult getCurRecNum(hDBICur tabH,
                       UINT32 *recNum)
{
    DBIResult    lastError;
    CURProps     curProps;
    RECProps     recProps;

    lastError = DbiGetCursorProps(tabH,
                                  &curProps);
    if (lastError != DBIERR_NONE)
    {
        return lastError;
    }

    if (!strcmp(curProps.szTableType,
                szPARADOX))
    {
        lastError = DbiGetSeqNo(tabH,
                                (pUINT32)recNum);
    }
    else if (!strcmp(curProps.szTableType,
                     szDBASE))
    {
        lastError = DbiGetRecord(tabH,
                                 dbiNOLOCK,
                                 NULL,
                                 &recProps);
        *recNum = recProps.iPhyRecNum;
```

```
                    }
                    else
                    {
                        lastError = DBIERR_NOTSUPPORTED;
                    }

                    return lastError;
                }
```

PXRecPrev() — DbiGetPriorRecord(tabH, dbiNOLOCK, NULL, NULL);

PXRecUpdate() — DbiModifyRecord(tabH, recH, TRUE);

PXSave() — DbiForceReread(tabH);

PXSetDefaults() — DbiOpenCfgInfoList()

PXSetHWHandler() — Not Supported

PXSrchFld() — DbiSetToKey(tabH, keySEARCHEQ, FALSE, iFlds iLen, pIdxBuf);

PXSrchKey() — DbiSwitchToIndex(&tabH, NULL, NULL, 0, TRUE);
DbiSetToKey(tabH, keySEARCHEQ, FALSE, iFlds iLen, pIdxBuf);
(can only search on the currently active index)

PXTblAdd() — DbiBatchMove(NULL, hSrcCur, NULL, hDestCur, batAPPEND, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, FALSE, NULL, TRUE);

PXTblClose() — DbiCloseCursor(&tabH);

PXTblCopy() — DbiCopyTable(hDb, TRUE, szSrcName, szPARADOX, szDestName);

PXTblCreate() — DbiCreateTable(hDb, TRUE, &TblDesc);

PXTblCreateMode() — DbiCreateTable(hDb, TRUE, &TblDesc);
(Use the optional parameters to change the level of the table.)

PXTblDecrypt() — Only Supported for Paradox Tables
Example for use with 'C':

```
DBIResult decryptTable(hDBIDb hDb,
                       const char *tableName)
{
    CRTblDesc     crTblDesc;
    DBIResult     lastError;

    // Clear the buffer
```

```
    memset(&crTblDesc, 0,
            sizeof(CRTblDesc));

    // name of the table
    strcpy(crTblDesc.szTblName, tableName);

    lastError = DbiDoRestructure(hDb, 1,
                                 &crTblDesc,
                                 NULL, NULL,
                                 NULL,
                                 FALSE);

    return lastError;
}
```

PXTblDelete()

```
DbiDeleteTable(hDb, szTblName, szPARADOX);
```

PXTblEmpty()

```
DbiEmptyTable(hDb, tabH, szTblName,
              szPARADOX);
```

PXTblEncrypt()

(Only Supported for Paradox Tables)
Example for use with 'C':

```
DBIResult encryptTable(hDBIDb hDb,
                       const char *tableName,
                       const char *password)
{
    CRTblDesc    crTblDesc;
    DBIResult    lastError;

    // Clear the buffer
    memset(&crTblDesc, 0,
            sizeof(CRTblDesc));

    // name of the table
    strcpy(crTblDesc.szTblName, tableName);
    // Master password suplied for the table
    crTblDesc.bProtected = TRUE;
    // Password for the table
    strcpy(crTblDesc.szPassword, password);

    lastError = DbiDoRestructure(hDb, 1,
                                 &crTblDesc,
                                 NULL, NULL,
                                 NULL, FALSE);

    return lastError;
}
```

PXTblExist()

Not directly supported. Need to use DbiOpenTableList to get a list of available tables, and then search that list for the table. Example for use with 'C':

```
BOOL tableExists(hDBIDb hDb,
                 const char *userName,
                 const char *tableName,
                 const char *tableType)
```

```
{
    BOOL            exists = FALSE;
    BOOL            isLocal = TRUE;
    hDBICur         tabH    = 0;
    char           *tblName;
    TBLBaseDesc tblDesc;
    CHAR            remoteName[DBIMAXNAMELEN+1];
    DBIResult    lastError;

    if ((!strcmp(tableType, szPARADOX)) ||
        (!strcmp(tableType, szDBASE)) ||
        (!strcmp(tableType, szASCII)))
    {
        tblName = _fstrtok((pCHAR)tableName,
                           ".");
        isLocal = TRUE;
    }
    else if ((!strcmp(tableType, "ORACLE"))
                ||
             (!strcmp(tableType, "SYBASE")))
    {
        strcpy(remoteName, userName);
        strcat(remoteName, ".");
        strcat(remoteName, tableName);
        tblName = (pCHAR)tableName;
        isLocal = FALSE;
    }
    else
    {
        strcpy(remoteName, tableName);
        tblName = (pCHAR)tableName;
        isLocal = FALSE;
    }

    if (tblName == NULL)
    {
        lastError = DBIERR_INVALIDTABLENAME;
        return FALSE;
    }

    lastError = DbiOpenTableList(hDb, FALSE,
                                 TRUE,"*.*",
                                 &tabH);
    if (lastError != DBIERR_NONE)
    {
        return FALSE;
    }

    lastError = DbiSetToBegin(tabH);
    if (lastError != DBIERR_NONE)
    {
        return FALSE;
    }

    while ((DbiGetNextRecord(tabH,dbiNOLOCK,
                             (pBYTE)&tblDesc,
```

```
                                        NULL))
                    == DBIERR_NONE)
            {
                if (!stricmp(tblDesc.szName,
                            tblName))
                {
                    // Check if the types match
                    if (strcmp(tblDesc.szType,
                                tableName) &&
                        (isLocal))
                    {
                        // keep searching it the
                        // table is of the wrong
                        // type
                        continue;
                    }
                    lastError = DBIERR_NONE;
                    DbiCloseCursor(&tabH);
                    return TRUE;
                }
            }

            if (tabH)
            {
                DbiCloseCursor(&tabH);
            }

            lastError = DBIERR_NOSUCHTABLE;

            return exists;
        }
```

PXTblMaxSize()

```
    DbiCreateTable(hDb, TRUE, &TblDesc);
```

(This is done using the optional parameters in the table descriptor)

PXTblName()

```
    DbiGetCursorProps(tabH, &CurProps);
strcpy(szTblName, CurProps.szName);
```

PXTblNRecs()

```
    DbiGetRecordCount(tabH, &iRecCount);
```

(This function returns an approximation of the number of records in the table.)

PXTblOpen()

```
    DbiOpenTable(hDb, szTblName, szPARADOX,
                NULL, NULL, indexID,
                dbiREADWRITE, dbiOPENSHARED,
                xltFIELD, TRUE, NULL, &tabH);
```

PXTblProtected()

Not directly supported. For local tables, need to call DbiOpenTableList and get the information for that table.
Example for use with 'C':

```
BOOL isProtected(hDBIDb hDb,
                    const char *tableName,
                    const char *tableType)
{
```

```
BOOL            protect = FALSE;
hDBICur         tabH      = 0;
char            *tblName;
TBLFullDesc     tblDesc;
DBIResult       lastError;

// Different methodology required for
// local and remote tables
if ((!strcmp(tableType, szPARADOX)) ||
     (!strcmp(tableType, szDBASE)) ||
     (!strcmp(tableType, szASCII)))
{
     tblName = _fstrtok((pCHAR)tableName,
                        ".");
     if (tblName == NULL)
     {
          lastError =
                DBIERR_INVALIDTABLENAME;
          return FALSE;
     }

     lastError = DbiOpenTableList(hDb,
                                  TRUE, TRUE,
                                  "*.*", &tabH);
     if (lastError != DBIERR_NONE)
     {
          return FALSE;
     }

     lastError = DbiSetToBegin(tabH);
     if (lastError != DBIERR_NONE)
     {
          return FALSE;
     }

     while ((DbiGetNextRecord(tabH,
                              dbiNOLOCK,
                              (pBYTE)&tblDesc,
                              NULL))
               == DBIERR_NONE)
     {
          if (!stricmp(
                    tblDesc.tblBase.szName,
                    tblName))
          {
               if (stricmp(
                    tblDesc.tblBase.szType,
                    tableType))
               {
                    // Keep searching if the
                    // type doesn't match
                    continue;
               }
               if (
                   tblDesc.tblExt.bProtected
                   == TRUE)
```

```
                    {
                        // Table is protected
                        lastError = DBIERR_NONE;
                        DbiCloseCursor(&tabH);
                        return TRUE;
                    }
                    else
                    {
                        lastError = DBIERR_NONE;
                        DbiCloseCursor(&tabH);
                        return FALSE;
                    }
                }
            }

            if (tabH)
            {
                DbiCloseCursor(&tabH);
            }

            lastError = DBIERR_NOSUCHTABLE;
        }
        else
        {
            lastError = DbiOpenTable(hDb,
                                (pCHAR)tableName,
                                (pCHAR)tableType,
                                NULL, NULL, NULL,
                                dbiREADONLY,
                                dbiOPENSHARED,
                                xltNONE, FALSE,
                                NULL, &tabH);
            if (lastError == DBIERR_NONE)
            {
                DbiCloseCursor(&tabH);
                protect = FALSE;
                return protect;
            }
            else if (lastError ==
                        DBIERR_INVALIDPASSWORD)
            {
                lastError = DBIERR_NONE;
                protect = TRUE;
                return protect;
            }
        }

        return protect;
    }
```

PXTblRename()

```
DbiRenameTable(hDb, szOldName, szTableType,
                szNewName);
```

PXTblUpgrade()

```
DbiDoRestructure(hDb, 1, pTblDesc, pSaveAs,
                    NULL, NULL, FALSE);
```

(Need to use the Optional Parameters to set
the level of the table. Local tables only.)

PXWinInit() DbiInit(NULL);
DbiOpenDatabase(NULL, NULL, dbiREADWRITE,
dbiOPENSHARED, NULL, 0,
NULL, NULL, &hDb);

Making your own hotkeys.

NUMBER : 2659
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Making your own hotkeys.

Q: How can I trap for my own hotkeys?

A: First: set the form's KeyPreview := true;

Then, you do something like this:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if (ssCtrl in Shift) and (chr(Key) in ['A', 'a']) then
    ShowMessage('Ctrl-A');
end;
```

How to scroll your form with pgUP and pgDn.

NUMBER : 2661
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to scroll your form with pgUP and pgDn.

Q. How can you do scrolling functions in a TForm component using keyboard commands? For example, scrolling up and down when a PgUp or PgDown is pressed. Is there some simple way to do this or does it have to be programmed by capturing the keystrokes and manually responding to them?

A. Form scrolling is accomplished by modifying the VertScrollbar or HorzScrollbar Postion properties of the form. The following code demonstrates how to do this:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

Introduction to BDE Programming

NUMBER : 2761
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Introduction to BDE Programming

Introduction to BDE Programming
BDE v2.0

I) Project Setup
This section covers the basics of what needs to be done in order to set up a Borland Database Engine project or makefile.

II) Design Overview
A high level overview of the steps that are required to create a simple application which retrieves fields from a table

III) Design Specifics
Detailed description, with code examples, of the steps described in part II.

IV) Error Handler
This section contains the error handler used by this example.

I) Project Setup

This document provides a quick introduction to programming with the Borland Database Engine. By the end of this section, you should have a simple EasyWin, BDE example which gets a record from a table and displays the first two fields.

The following steps must be followed when starting to write a BDE application.

First, set up the project or makefile
MAIN.CPP - File to contain your code
DBERR.CPP - Error handling routine
IDAPI.LIB - BDE Import Library
MAIN.DEF - Module Definition file

Select the large memory model.

For this simple application, set the target to be an EasyWin application. This way we don't have to deal with any Windows UI issues.

It is also suggested to install the IDAPI.TOK file so as to have syntax hilighting for BDE functions and types (directions on this are included in the \BDE\DOC\IDAPITOK.TXT file).

The on-line help file, IDAPI.HLP, can also be incorporated into the BC 4.5 OpenHelp architecture, allowing context sensitive help on IDAPI types and functions.

Other required setup:

The Borland Database Engine is fairly stack intensive, especially when doing batch operations and queries, so it is generally recommended that the stack be set to a minimum of 25k (for Windows C/C++ applications, the stack size is set in the module definition file)

Increase the number of file handles available to the application. By default, the BDE assumes that it has access to 48 physical file handles (set by the BDE Configuration utility, System page | MAXFILEHANDLES option, 5-255). Due to this default setting, it is usually best to set the number of file handles available to an application to a minimum of 68 using the Windows API function SetHandleCount. Seemingly random errors can occur when the BDE does not have access to enough file handles.

Use the Debug Layer when developing applications. The debug layer performs much stricter error checking than the regular DLL, resulting in fewer GP faults and less re-booting of your machine. It will also produce a trace output of which IDAPI functions were called by an application. Note that use of the debug layer requires the use of both the Debug DLL (Set using the DLLSwap utility), as well as a call to the DbiDebugLayerOptions function.

Make certain to compile with 'Allocate enums as ints' selected (in the BC 4.x IDE, Options | Project | Compiler | Code Generation). A number of structures, such as CURProps, make use of enumerations. Within the DLL, these are allocated as two byte values. Turning off this option will result in your code passing only one byte. This error generally manifests itself with stack corruption problems, such as GP faults when calling or returning from a function.

Within a module to contain BDE code, include the following header files:
WINDOWS.H
IDAPI.H

Note that order is important. The WINDOWS.H file must be included before IDAPI.H.

II) Design Overview
==========

Before we start writing code, a brief overview of what needs to be done to get a record from a tables:

Increase the number of file handles available to the application
Initialize the Borland Database Engine
Enable the debug layer
Open a database object
Set the database object to point to the directory containing the table
Set the directory for temporary objects
Open a table, creating a cursor object
Get the properties of the table
Using these properties, allocate memory for a record buffer
Position the cursor on the desired record
Get the desired record from the cursor (table)
Get the desired field(s) from the record
Free all resources

Note that throughout the following short example unfamiliar variable types may be used. These are IDAPI variable types that are defined in the IDAPI.H header file. For example, BYTE, BOOL, CHAR, and FLOAT.

III) Design Specifics

Increase the number of file handles available to the application

A call to the Windows API function SetHandleCount will increase the number of file handles available to an application in the Windows environment:

```
int      HandlesAvail;
int      HandlesWanted;

HandlesAvail = 0;
HandlesWanted = 68;

HandlesAvail = SetHandleCount(HandlesWanted);
if (HandlesAvail < HandlesWanted)
{
     // Display message re: not enough available
     // file handles
     return;
}
```

Note that in non-trivial cases it is recommended to determine the number of file handles that are specified in the BDE configuration file. This would be done using the DbiOpenCfgInfoList function, which will be described in a later section on configuration management.

Initialize the Borland Database Engine

The Borland Database Engine is initialized using the DbiInit function:

```
CHKERR(DbiInit(NULL));
```

Note that the CHKERR macro is defined in the DBERR.H file, which is a part of the Error Handling module. The call to DbiInit allocates system resources for the client.

Enable the Debug layer

The following code is used to enable the debug layer, outputting trace information to a text file on disk.

```
DbiDebugLayerOptions(DEBUGON | OUTPUTTOFILE, "MYTRACE.INF");
```

Note that in certain situations, you may also want to use the FLUSHEVERYOP flag, which will force output to the trace file after every operation. While this is slower, it is useful when a GP fault occurs....

Open a database object

The next step that is needed is to open a database object. All table access must be performed within the context of a database. Local databases generally use what is referred to as the "STANDARD" database, which is what we will be using in this example. The preferred method would be to create an alias to a local directory and using that as the database. This allows for easy modification in the future if one day it is decided to move the application from using dBASE tables to using InterBase tables. The DbiOpenDatabase function is used to open a database:

```
hDBIDb     hDb;      // Handle to the Database

hDb = 0; // Initialize to zero for cleanup
          // purposes

CHKERR_CLEANUP(
     DbiOpenDatabase(NULL, // Database name
                              // - NULL for standard Database
                         NULL, // Database type -
                              // NULL for standard Database
                         dbiREADWRITE, // Open mode - Read/Write or
                              // Read only
                         dbiOPENSHARED, // Share mode - Shared or
                              // Exclusive
                         NULL, // Password - not needed for the
                              // STANDARD database
                         NULL, // Number of Optional Parameters
                         NULL, // Field Desc for Optional
                              // Parameters
                         NULL, // Values for the
                              // Optional Parameters
```

```
                        &hDb) // Handle to the database
                       );

// At the end of the function
CleanUp:
    // Close only if open
    if (hDb != 0)
    {
        DbiCloseDatabase(&hDb);
    }
```

Set the database object to point to the directory containing
---
the table

The working directory next needs to be set to the directory containing the table. While it is possible to open a table in other directories, specifying the absolute path, it is generally recommended to open tables in the working directory, as a number of operations, such as getting a list of available tables, work off the current directory. The DbiSetDirectory function is used to set the working directory (using the default location of the BDE example tables):

```
CHKERR_CLEANUP(
    DbiSetDirectory(hDb, // Handle to the
                         // database which is being modified
                    "C:\\BDE\\EXAMPLES\\TABLES")
                         // The new working directory
                   );
```

Note that the full, absolute path needs to be used. Relative paths are not supported.

Set the directory for temporary objects

While not all BDE applications create temporary objects, but larger applications will at one time or another. For example, the result set from a query or the records which cause a key violation in a restructure will be placed in a temporary table. By default, this temporary, or private directory, as it is called, is the startup directory. This will cause a problem if the application is run off a network or a CD-ROM., as the directory cannot be shared, and it must we writable. The DbiSetPrivateDir function is used to set the private directory for a client:

```
CHKERR_CLEANUP(
    DbiSetPrivateDir("c:\\<SOMEDIR>")
      //  Select a directory on a local drive that is not
      //  used by other applications.
                    );
```

Note that the full, absolute path needs to be used. Relative

paths are not supported.

Open a table, creating a cursor object

Next, open the table. Upon opening a table, a cursor object is created and returned to the calling application. A cursor object is basically an abstraction which allows queries and tables to be accessed in the same method:

```
hDBICur hCur;      // Handle to the cursor (table)
CHAR      szTblName[DBIMAXNAMELEN]; // Table name - DBIMAXNAMELEN
                  // is defined in IDAPI.H
CHAR      szTblType[DBIMAXNAMELEN]; // Table Type

hCur = 0; // Initialize to zero for cleanup
            // purposes

strcpy(szTblName, "customer"); // Name of the table
strcpy(szTblType, szPARADOX);   // Type of the tables
                                 // - szPARADOX is defined in IDAPI.H

CHKERR_CLEANUP(
     DbiOpenTable(hDb,    // Handle to the
                          // database to which this cursor will
                          // be related
                  szTblName, // Name of the table
                  szTblType, // Type of the table - only
                          // used for local tables
                  NULL, // Index Name - Optional
                  NULL, // IndexTagName - Optional.
                          // Only used by dBASE
                  0,      // IndexId - 0 = Primary. Optional.
                          // Paradox and SQL only
                  dbiREADWRITE, // Open Mode -
                          // Read/Write or Read Only
                  dbiOPENSHARED, // Shared mode -
                          // SHARED or EXCL
                  xltFIELD, // Translate mode
                          // FIELD or NONE
                          // FIELD:   Convert data from table
                          //     format to C format
                          // NONE: Leave data in it's native
                          // state
                  FALSE,// Unidirectional -
                          // False means can navigate forward
                          // and back.
                  NULL, // Optional Parameters.
                  &hCur)// Handle to the cursor
                 );

CleanUp:
     // Close only if open
     if (hCur != 0)
     {
          DbiCloseCursor(&hCur); // Note the use of DbiCloseCursor
```

```
                                        // - there is no DbiCloseTable.
    }
```

Get the properties of the table

We next need to determine the size of the record buffer for the table. This information is gotten from the cursor via the DbiGetCursorProps function. The Cursor properties include information on the table name, size, type, number of fields, and record buffer size. More information can be found on cursor properties in the on-line help under CURProps.

```
CURProps      curProps; // Properties of the cursor

CHKERR_CLEANUP(
    DbiGetCursorProps(hCur,         // Handle to the cursor
                      &curProps) // Properties of the cursor
                                 // (table)
                );
```

curProps.iRecBufSize contains the size of the record buffer.

Using these properties, allocate memory for a record buffer

```
pBYTE          pRecBuf; // Pointer to the record buffer

pRecBuf = NULL; // For cleanup purposes

// Allocate memory for the record buffer
pRecBuf = (pBYTE)malloc(curProps.iRecBufSize * sizeof(BYTE));
if (pRecBuf == NULL)
{
    // Display some error message
    goto CleanUp;
}

CleanUp:
    if (pRecBuf)
    {
        free(pRecBuf);
    }
```

Position the cursor on the desired record

The DbiSetToBegin() function is used to position the cursor to the "crack" before the first record in the table. Crack semantics basically allow for the current location of the cursor to be before the first record, between records, or after the last record. One of the reasons for having crack semantics is to allow the use of one function to access all records in a table. For example, in place of having to use DbiGetRecord the first time, and DbiGetNextRecord each subsequent time, it is now possible to use DbiGetNextRecord to get all records in a table.

```
CHKERR_CLEANUP(
    DbiSetToBegin(hCur) // Position the specified cursor
                        // to the crack before the first record
                );
```

Get the desired record from the cursor (table)

The DbiGetNextRecord function is normally used to get a record from a table. Note that the current record of the cursor is set to the record returned by this function (the next record in the table):

```
CHKERR_CLEANUP(
    DbiGetNextRecord(hCur,         // Cursor from which to get the
                                   // record.
                     dbiNOLOCK, // Lock Type
                     pRecBuf,     // Buffer to store the record
                     NULL)         // Record properties -
                                   // don't need in this case
                    );
```

Get the desired field(s) from the record

The last step is to get the field values out of the record buffer and into some local variable. Note that in this case we are making assumptions about which field is at which ordinal position within the table, as well as the size of the field. In general, it is recommended to use DbiGetFieldDescs to get information about a field before retrieving it. Also note that a single function, DbiGetField, is used to get all fields, other than BLOBs, from a table.

```
FLOAT         custNum;
BOOL          isBlank;

CHKERR_CLEANUP(
    DbiGetField(hCur,      // Cursor which contains the record
                1,         // Field Number of the "Customer" field.
                pRecBuf, // Buffer containing the record
                (pBYTE)&custNum, // Variable for the Customer
                         // Number
                isBlank) // Is the field blank?
            );
```

Free all resources

After all desired operations have been performed, the resources allocated on behalf of the application need to be cleaned up. In addition to any memory explicitly allocated using malloc or new, all engine objects must also be cleaned up, including the cursor, database, and engine:

```
CleanUp:
```

```
    if (pRecBuf)
    {
        // Free the record buffer
        free(pRecBuf);
    }
    if (hCur !=0)
    {
        // Close the cursor
        DbiCloseCursor(&hCur);
    }
    // Close only if open
    if (hDb != 0)
    {
        // Close the database
        DbiCloseDatabase(&hDb);
    }

    // Close the BDE.
    DbiExit();
```

IV) Error handler

```
// DBERR.H

// This file contains macros to use around BDE functions
// which are not expected to fail, or which would
// result in a fatal error. More explicit error handling needs
// to be done in the application when a non-fatal
// error can occur, for example, getting an End of file error,
// DBIERR_EOF, from a DbiGetNextRecord
// loop.

#ifndef __DBERR_H
#define __DBERR_H

#include <idapi.h>

// Macro to use when no cleanup is required

#define CHKERR(parm) DBIError(__FILE__, __LINE__, \
                              #parm, parm) ; \
                              if (GlobalDBIErr) { \
                                  return GlobalDBIErr ;}

// Macro to use when cleanup must be done within a function
// for example, to close open tables or to free allocated memory.
// Note: must have a CleanUp label defined in each function
// where this macro is used. The CleanUp
// label should start the section of a function which frees the
// resources allocated within a given function

#define CHKERR_CLEANUP(parm) DBIError(__FILE__, __LINE__, \
                              #parm, parm) ; \
                              if (GlobalDBIErr) { \
                                  goto CleanUp ;}
```

```
// Global variable to contain the result code
extern DBIResult GlobalDBIErr;

// Prototype for the error handling function
DBIResult DBIError(pCHAR, UINT16, pCHAR, DBIResult);

#endif


// DBERR.C

#include <string.h>
#include "DBERR.h"

//      The way to use this macro is to include the DBERR.H.
//      Then pass an IDAPI function as a parameter to the
//      macro:
//
//      #define CHKERR(parm) DBIError(__FILE__, __LINE__, \
//                                    #parm, parm) ; \
//                                    if ( GlobalDBIErr ) { \
//                                         return GlobalDBIErr ;}
//
//      You would then use it as such:
//           CHKERR(DbiCreateTable(hDb, bOverWrite,
//                                 &crTblDsc)) ;

// Global variable to hold return value from BDE functions
DBIResult GlobalDBIErr;

// Global variables to contain the error messages. Defined
// globally to ensure that an error message can be displayed
// even if the system is out of memory (and to keep it off
// the stack)

// Contains just the BDE error message
static char szDBIStatus[(DBIMAXMSGLEN * 7)+1];

// Contains the entire message, including file name and line
// number of the offending function.
static char szMessage[(DBIMAXMSGLEN * 7)+1+110];

//=========================================
//  Function:
//           DBIError();
//
//  Input:   module name (pCHAR), line number (UINT16),
//               Engine function name (pCHAR),
//               Result (DBIResult)
//
//  Return: A DBIResult value.
//
//  Description:
//           This is a function which takes in the information of
//           where the error occurred and displays error
//           information in a message box.
```

```
//============================================================
DBIResult
DBIError (pCHAR module, UINT16 line, pCHAR function,
          DBIResult retVal)
{
    DBIErrInfo   ErrInfo; // Structure to contain the error
                          // information

    if (retVal == DBIERR_NONE)
    {
        GlobalDBIErr = DBIERR_NONE;
        return retVal;
    }
    // Don't want to call functions if the DLL is not there....
    if (retVal != DBIERR_CANTFINDODAPI)
    {
        // Get as much error information as possible
        DbiGetErrorInfo(TRUE, &ErrInfo);

        // Make certain information is returned on the correct
        // error (that this function was called
        // immediately after the function that caused the error.
        if (ErrInfo.iError == retVal)
        {
            strcpy(szDBIStatus, ErrInfo.szErrCode);

            if (strcmp(ErrInfo.szContext1, ""))
            {
                strcat(szDBIStatus, ErrInfo.szContext1);
            }
            if (strcmp(ErrInfo.szContext2, ""))
            {
                strcat(szDBIStatus, ErrInfo.szContext2);
            }
            if (strcmp(ErrInfo.szContext3, ""))
            {
                strcat(szDBIStatus, ErrInfo.szContext3);
            }
            if (strcmp(ErrInfo.szContext4, ""))
            {
                strcat(szDBIStatus, ErrInfo.szContext4);
            }
        }
        else {
            DbiGetErrorString(retVal, szDBIStatus);
        }

        sprintf(szMessage,
                "Module:\t\t%s\nFunction:\t%s\nLine:\t\t%d\n"
                "Category:\t%d\nCode:\t\t%d\nError:\r\n\r\n%s\n",
                module, function, line, ErrCat(retVal),
                ErrCode(retVal), szDBIStatus);

        MessageBox(NULL, szMessage, "BDE Error",
                   MB_ICONEXCLAMATION);
    }
```

```
    else
    {
        // Display an error message if the BDE DLL's cannot be
        // found.
        MessageBox(NULL, "Cannot find Borland Database"
                    " Engine files: Check the [IDAPI] section"
                    " in WIN.INI.",   "BDE Initialization Error",
                    MB_ICONHAND | MB_OK);
    }

    GlobalDBIErr = retVal;
    return retVal;
}
```

New Language Features in Delphi 2.0 - 32 Bit

NUMBER : 2695
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : February 28, 1996

TITLE : New Language Features in Delphi 2.0 - 32 Bit

New Language Features in Delphi 2.0 - 32

Delphi32 defines several new data types that reduce the limitation set by Windows 3.1. Delphi32 has also changed a few data types to take advantage of the 32 bit environment.

New data types include:

- Character type
- String type
- Variant type
- Currency type

Changed data types:

- Integer
- Cardinal

Additional Syntax:

- Unit finalization section

New Data Types
--- ---- -----

Character Type

Delphi 2.0 introduces new wide character types to support Unicode. Delphi 1.0 treated characters as 8-bit values of type Char.

These are the standard types that represent characters in Delphi32.

ANSIChar - A standard 8-bit ANSI character, equivalent to the Char type in previous versions of Delphi.

WideChar - A 16-bit character, representing a Unicode character. If the high-order byte is zero, the low-order byte contains an ANSI character.

Char - By default, Char is equivalent to ANSIChar. Char works in the same way as the implementation-dependent Integer type, which is equivalent to SmallInt in 16-bit versions of Delphi and to LongInt in 32-bit versions of Delphi. In Delphi 2.0, Char defaults to

an 8-bit value.

Character-pointer types:

Pointer type	Character type
PANSIChar	ANSIChar
PWideChar	WideChar
PChar	Char

The semantics of all the character-pointer types are identical. The only thing that varies is the size of the character pointed to.

String Type

Delphi 2.0 supports strings of nearly unlimited length in addition to the 255-character counted strings previously supported. A new compiler directive, $H, controls whether the reserved word "string" represents a short string or the new, long string. The default state of $H, is $H+, using long strings by default. Alll Delphi 2.0 components use the new long string type.

These are the new string types.

ShortString - A counted string with a maximum length of 255 characters. Equivalent to string in Delphi 1.0. Each element is of type ANSIChar.

AnsiString - A new-style string of variable length, also called a "long string." Each element is of type ANSIChar.

string - Either a short string or an ANSI string, depending on the value of the $H compiler directive.

Here are the compatibility issues.

Although most string code works interchangeably between short strings and long strings, there are certain short-string operations that either won't work on long strings at all or which operate more efficiently when done a different way. The following table summarizes these changes.

Short String operation	Long string equivalent	Explanation
PString type	string	All long strings are dynamically allocated, so PString is redundant and requires more bookkeeping.
S[0] := L	SetLength(S,L) SetString(S,P,L)	Because long strings are dynamically allocated, you must call the SetLength procedure to allocate the appropriate amount of memory.
StrPCopy (Buffer, S)	PChar(S)	You can typecast long strings into null-terminated strings. The address of the long string is the

		address of its first character, and the long string is followed by a null.
S := StrPas(P)	S := P	Long strings can automatically copy from null-terminated strings.

Long strings cannot be passed to OpenString-type parameters or var short-string parameters.

Varient Type

Delphi 2.0 introduces variant types to give you the flexibility to dynamically change the type of a variable. This is useful when implementing OLE automation or certain kinds of database operations where the parameter types on the server are unknown to your Delphi-built client application.

A variant type is a 16-byte structure that has type information embedded in it along with its value, which can represent a string, integer, or floating-point value. The compiler recognizes the standard type identifier Variant as the declaration of a variant.

In cases where the type of a variant is incompatible with the type needed to complete an operation, the variant will automatically promote its value to a compatible value, if possible. For instance, if a variant contains an integer and you assign it to a string, the variant converts its value into the string representing the integer number, which is then assigned to the string.

You can also assign a variant expression to a variable of a standard type or pass the variant as a parameter to a routine that expects a standard type as a parameter. Delphi coerces the variant value into a compatible type if necessary, and raises an exception if it cannot create a compatible value.

Currency Type

Delphi 2.0 defines a new type called Currency, which is a floating-point type specifically designed to handle large values with great precision. Currency is assignment-compatible with all other floating-point types (and variant types), but is actually stored in a 64-bit integer value much like the Comp type.

Currency-type values have a four-decimal-place precision. That is, the floating-point value is stored in the integer format with the four least significant digits implicitly representing four decimal places.

Changed Data Types
------- ---- -----
The implementation-dependent types Integer and Cardinal are 32-bit values in Delphi 2.0, where they were 16-bit values in Delphi 1.0. To explicitly declare 16-bit integer data types, use the SmallInt and Word types.

Additional Syntax
---------- ------
You can include an optional finalization section in a unit. Finalization is the counterpart of initialization, and takes place when the application shuts down. You can think of the finalization section as "exit code" for a unit. The finalization section corresponds to calls to ExitProc and AddExitProc in Delphi 1.0.

The finalization begins with the reserved word finalization. The finalization section must appear after the initialization section, but before the final end. statement.

Once execution enters an initialization section of a unit, the corresponding finalization section is guaranteed to execute when the application shuts down. Finalization sections must handle partially-initialized data properly, just as class destructors must.

Finalization sections execute in the opposite order that units were initialized. For example, if your application initializes units A, B, and C, in that order, it will finalize them in the order C, B, and A.

The outline for a unit therefore looks like this:

```
unit UnitName;
interface
{ uses clause; optional }
...
implementation
{ uses clause; optional }
...
initialization { optional }
...
finalization { optional }
...
end.
```

How to Circumvent the "index not found" Exception.

NUMBER : 2711
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : December 13, 1995

TITLE : How to Circumvent the "index not found" Exception.

Q: How do I open a dBASE table without the required MDX file?
I keep getting an "Index not found..." exception.

A: When you create a dBASE table with a production index (MDX), a special byte is set in the header of the DBF file. When you subsequently attempt to re-open the table, the dBASE driver will read that special byte, and if it is set, it will also attempt to open the MDX file. When the MDX file cannot be opened, an exception is raised.

To work around this problem, you need to reset the byte (byte 28 decimal) in the DBF file that causes the MDX dependency to zero.

The following unit is a simple example of how to handle the exeption on the table open, reset the byte in the DBF file, and re-open the table.

```
unit Fixit;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, DB, DBTables, Grids, DBGrids;

type
  TForm1 = class(TForm)
    Table1: TTable;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

const
```

```
  TheTableDir = 'c:\temp\';
  TheTableName = 'animals.dbf';

procedure RemoveMDXByte(dbFile: String);
{ This procedure accepts a DBF file as a parameter.   It will patch}
{ the DBF header, so that it no longer requires the MDX file }
const
  Value: Byte = 0;
var
  F: File of byte;
begin
  AssignFile(F, dbFile);
  Reset(F);
  Seek(F, 28);
  Write(F, Value);
  CloseFile(F);
end;

procedure TForm1.Button1Click(Sender: TObject);
{ This procedure is called in response to a button click.   It      }
{ attempts to open a table, and, if it can't find the .MDX file, }
{ it patches the DBF file and re-execute the procedure to           }
{ re-open the table without the MDX   }
begin
  try
    { set the directory for the table }
    Table1.DatabaseName := TheTableDir;
    { set the table name }
    Table1.TableName := TheTableName;
    { attempt to open the table }
    Table1.Open;
  except
    on E:EDBEngineError do
      { The following message indicates the MDX wasn't found: }
      if Pos('Index does not exist. File', E.Message)>0 then begin
        { Tell user what's going on. }
        MessageDlg('MDX file not found.   Attempting to open
                    without index.', mtWarning, [mbOk], 0);
        { remove the MDX byte from the table header }
        RemoveMDXByte(TheTableDir + TheTableName);
        { Send the button a message to make it think it was }
        { pressed again.   Doing so will cause this procedure to }
        { execute again, and the table will be opened without }
        { the MDX }
        PostMessage(Button1.Handle, cn_Command, bn_Clicked, 0);
      end;
  end;
end;

end.
```

College Student Guide to Reading and Writing Files

NUMBER : 2719
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : College Student Guide to Reading and Writing Files

College Student Crib Notes to:

1) Printing out your programs' output to a file.
2) Printing out your programs' output to the printer.
3) Reading from an input file.

Printing Out Your Programs Output to a File

Often times, professors will require more than your honesty and good faith to receive full credit for your program. They will want to see both your program listing and the output generated from the program. But how do you do this in Delphi and our Pascal Products ???

Simply in Delphi ....

```
program CrtApp;
uses WinCrt;
var outfile: TextFile;
begin
  AssignFile(outfile, 'c:\outfile.txt');
  Rewrite(outfile);
  writeln(outfile, 'Hello World from Delphi');
  writeln(outfile, 'My Program works, and here is ' +
                   'the output to prove it...');
  CloseFile(outfile);
end.
```

Simply in Pascal .....

```
Program HelloWorld;
var
  outfile: text;
begin
  assign(outfile, 'c:\output.txt');
  rewrite(outfile);
  writeln(outfile, 'Hello World');
  writeln(outfile, 'My Program works, and here is the
          output to prove it...');
  close(outfile);
end.
```

Printing Out Your Programs' Output to the Printer

-------- --- ---- --------- ------ -- --- -------

In some cases, there may be a need to print output directly to the printer. This is the way.

In Delphi ...

```
program CrtApp;
uses WinCrt;
var outfile: TextFile;
begin
  assignfile(outfile, 'LPT1');
  rewrite(outfile);
  writeln(outfile, 'Hello World from Delphi');
  writeln(outfile, 'My Program works, and here is ' +
                        'the output to prove it...');
  closefile(outfile);
end.
```

In Pascal ...

```
Program HelloWorld;
var
  outfile: text;
begin
  assign(outfile, 'LPT1');
  rewrite(outfile);
  writeln(outfile, 'Hello World');
  writeln(outfile, 'My Program works, and here is the
            output to prove it...');
  close(outfile);
end.
```

Reading From an Input File

------- ---- -- ----- ----

Many times, it will be neccessary to read information from an input file supplied by another person. This is an example of how to implement this.

In Delphi ...

```
program CrtApp;
uses WinCrt;
var
  infile, outfile: TextFile;
  num_lines, x: integer;
  line: string;
begin
  assignfile(infile, 'C:\INFILE.TXT');
  assignfile(outfile, 'C:\OUTFILE.TXT');
  reset(infile);   {reset moves the pointer to the beginning}
                      {of the file.                          }
  rewrite(outfile);   {clears the contents of the file}
  readln(infile, num_lines);
  for x:= 1 to num_lines do
  begin
```

```
        readln(infile, line);
        writeln(outfile, line);
      end;
      closefile(infile);
      closefile(outfile);
    end.
```

In Pascal ...

```
    Program ReadInput;
    var
      infile, outfile: text;
      num_lines, x: integer;
      line: string;
    begin
      assign(infile, 'C:\INFILE.TXT');
      assign(outfile, 'C:\OUTFILE.TXT');
      reset(infile);   {reset moves the pointer to the beginning}
                          {of the file.                          }
      rewrite(outfile);   {clears the contents of the file}
      readln(infile, num_lines);
      for x:= 1 to num_lines do
      begin
        readln(infile, line);
        writeln(outfile, line);
      end;
      close(infile);
      close(outfile);
    end.
```

```
{START INFILE.TXT}
2
Hello World
My Program works, and here is the output to prove it.
{END INFILE.TXT}
```

For further reference, please check your Programmer's Reference. Look for the topics (AssignFile, Assign, Reset, Rewrite, readln, writeln, Close, CloseFile)

This document was written and inspired by the author's sympathy for the starving college student-- only because he was recently in your shoes..!!

Some current internal limits of BDE

NUMBER : 2751
PRODUCT : BDE
VERSION : 2.x
OS : Windows
DATE : April 2, 1997

TITLE : Some current internal limits of BDE

Here are the maximum limits for some common BDE objects.

GENERAL LIMITS

48	Clients in system
32	Sessions per client
32	Open databases per session
32	Loaded drivers
64	Sessions in system
4000	Cursors per session
16	Entries in error stack
8	Table types per driver
16	Field types per driver
8	Index types per driver
48K	Size of configuration (IDAPI.CFG) file

PARADOX LIMITS

127	Tables open per system
64	Record locks on one table (16Bit)
255	Record locks on one table (32Bit)
255	Records in transactions on a table (32 Bit)
512	Open physical files (DB, PX, MB, X??, Y??, VAL, TV)
300	Users in one PDOXUSRS.NET file
255	Number of fields per table
255	Size of character fields
2	Billion records in a table
2	Billion bytes in .DB (Table) file
10800	Bytes per record for indexed tables
32750	Bytes per record for non-indexed tables
127	Number of secondary indexes per table
16	Number of fields in an index
255	Concurrent users per table
256	Megabytes of data per BLOb field
100	Passwords per session
15	Password length
63	Passwords per table
159	Fields with validity checks (32 Bit)
63	Fields with validity checks (16 Bit)

DBASE LIMITS

256	Open dBASE tables per system (16 Bit)
350	Open dBASE tables per system (32 Bit)
100	Record locks on one dBASE table (16 and 32 Bit)
100	Records in transactions on a dBASE table (32 Bit)
1	Billion records in a table
2	Billion bytes in .DBF (Table) file
4000	Size in bytes per record (dBASE 4)
32767	Size in bytes per record (dBASE for Windows)
255	Number of fields per table (dBASE 4)
1024	Number of fields per table (dBASE for Windows)
47	Number of index tags per .MDX file
254	Size of character fields
10	Open master indexes (.MDX) per table

Local SQL Reserved Words.

NUMBER : 2752
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Local SQL Reserved Words.

This file contains an alphabetical list of words reserved by
Local SQL in the Borland Database Engine. Note that this file
is provided as-is.

ACTIVE, ADD, ALL, AFTER, ALTER, AND, ANY, AS, ASC, ASCENDING,
AT, AUTO, AUTOINC, AVG

BASE_NAME, BEFORE, BEGIN, BETWEEN, BLOB, BOOLEAN, BOTH, BY,
BYTES

CACHE, CAST, CHAR, CHARACTER, CHECK, CHECK_POINT_LENGTH,
COLLATE, COLUMN, COMMIT, COMMITTED, COMPUTED, CONDITIONAL,
CONSTRAINT, CONTAINING, COUNT, CREATE, CSTRING, CURRENT,
CURSOR

DATABASE, DATE, DAY, DEBUG, DEC, DECIMAL, DECLARE, DEFAULT,
DELETE, DESC, DESCENDING, DISTINCT, DO, DOMAIN, DOUBLE, DROP

ELSE, END, ENTRY_POINT, ESCAPE, EXCEPTION, EXECUTE, EXISTS,
EXIT, EXTERNAL, EXTRACT

FILE, FILTER, FLOAT, FOR, FOREIGN, FROM, FULL, FUNCTION

GDSCODE, GENERATOR, GEN_ID, GRANT, GROUP,
GROUP_COMMIT_WAIT_TIME

HAVING, HOUR

IF, IN, INT, INACTIVE, INDEX, INNER, INPUT_TYPE, INSERT,
INTEGER, INTO, IS, ISOLATION

JOIN

KEY

LONG, LENGTH, LOGFILE, LOWER, LEADING, LEFT, LEVEL, LIKE,
LOG_BUFFER_SIZE

MANUAL, MAX, MAXIMUM_SEGMENT, MERGE, MESSAGE, MIN, MINUTE,
MODULE_NAME, MONEY, MONTH

NAMES, NATIONAL, NATURAL, NCHAR, NO, NOT, NULL,
NUM_LOG_BUFFERS, NUMERIC

OF, ON, ONLY, OPTION, OR, ORDER, OUTER, OUTPUT_TYPE, OVERFLOW

PAGE_SIZE, PAGE, PAGES, PARAMETER, PASSWORD, PLAN, POSITION, POST_EVENT, PRECISION, PROCEDURE, PROTECTED, PRIMARY, PRIVILEGES

RAW_PARTITIONS, RDB$DB_KEY, READ, REAL, RECORD_VERSION, REFERENCES, RESERV, RESERVING, RETAIN, RETURNING_VALUES, RETURNS, REVOKE, RIGHT, ROLLBACK

SECOND, SEGMENT, SELECT, SET, SHARED, SHADOW, SCHEMA, SINGULAR, SIZE, SMALLINT, SNAPSHOT, SOME, SORT, SQLCODE, STABILITY, STARTING, STARTS, STATISTICS, SUB_TYPE, SUBSTRING, SUM, SUSPEND

TABLE, THEN, TIME, TIMESTAMP, TIMEZONE_HOUR, TIMEZONE_MINUTE, TO, TRAILING, TRANSACTION, TRIGGER, TRIM

UNCOMMITTED, UNION, UNIQUE, UPDATE, UPPER, USER

VALUE, VALUES, VARCHAR, VARIABLE, VARYING, VIEW

WAIT, WHEN, WHERE, WHILE, WITH, WORK, WRITE

YEAR

OPERATORS:

||, -, *, /, <>, <, >, ,(comma), =, <=, >=, ~=, !=, ^=, (,)

Drag a form without clicking the caption bar

```
    NUMBER   :   2683
   PRODUCT   :   Delphi
   VERSION   :   All
         OS   :   Windows/Win32
      DATE    :   February 28, 1996

      TITLE   :   Drag a form without clicking the caption bar

{
Q: How can I make a form move by clicking and dragging in the client area
     instead of on the caption bar?

A: The easiest way to do this is to "fool" Windows into thinking that
     you're actually clicking on the caption bar of a form.    Do this by
     handling the wm_NCHitTest windows message as shown in the sample unit
     below.

     TIP: If you want a captioness borderless window similar to a floating
     toolbar, set the Form's Caption to an empty string, disable all of the
     BorderIcons, and set the BorderStyle to bsNone.
}

unit Dragmain;

interface

uses
   SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
   Forms, Dialogs, StdCtrls;

type
   TForm1 = class(TForm)
      Button1: TButton;
      procedure Button1Click(Sender: TObject);
   private
      procedure WMNCHitTest(var M: TWMNCHitTest); message wm_NCHitTest;
   end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.WMNCHitTest(var M: TWMNCHitTest);
begin
   inherited;                          { call the inherited message handler }
   if   M.Result = htClient then   { is the click in the client area?     }
      M.Result := htCaption;          { if so, make Windows think it's        }
                                      { on the caption bar.                   }
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;

end.

{ The text representation of the .DFM file is below:

object Form1: TForm1
  Left = 203
  Top = 94
  BorderIcons = []
  BorderStyle = bsNone
  ClientHeight = 273
  ClientWidth = 427
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 16
  object Button1: TButton
    Left = 160
    Top = 104
    Width = 89
    Height = 33
    Caption = 'Close'
    TabOrder = 0
    OnClick = Button1Click
  end
end

}
```

Comparison filters.

NUMBER : 2762
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Comparison filters.

```
/*

    This example shows how to set a filter on the first N
    characters of a character field. To use this example, backup
    the FILTER.C file in the BDE\EXAMPLES\SNIPIT directory. Copy
    this file to FILTER.C in that directory. Then compile and
    run SNIPIT, running the 'Cursor: Setting and Using filters'
    example.

*/

// BDE - (C) Copyright 1994 by Borland International

// Filter.c
#include "snipit.h"

static const char szTblName[] = "customer";
static const char szTblType[] = szPARADOX;

//============================================================
//   Function:
//              Filter();
//
//   Description:
//              This example shows how to use filters to limit the
//              result set of a table. Filters perform a function
//              similar to that of ranges, but more powerful
//              operations are supported.
//============================================================
void
Filter (void)
{
    hDBIDb          hDb;            // Handle to the database.
    hDBICur         hCur;           // Handle to the table.
    pBYTE           pcanExpr;       // Structure containing
                                    //    filter info.
    hDBIFilter      hFilter;        // Filter handle.
    UINT16          uSizeNodes;     // Size of the nodes in the
                                    //    tree.
    UINT16          uSizeCanExpr;   // Size of the header
                                    //    information.
    UINT16          uSizeLiterals;  // Size of the literals.
    UINT16          uTotalSize;     // Total size of the filter
                                    //    expression.
    UINT32          uNumRecs   = 10; // Number of records to
```

```
                                    //     display.
CANExpr             canExp;             // Contains the header
                                    //     information.
UINT16              Nodes[] =           // Nodes of the filter
                                    //     tree.
        {
            // Offset 0
            nodeCOMPARE,      // canCompare.nodeClass
            canEQ,              // canCompare.canOp
            1,                  // canCompare.bCaseInsensitive
            5,                  // canCompare.iParialLen (0 if
                                //     full length)
            12,                 // canBinary.iOperand1
            20,                 // canBinary.iOperand2
                                //     Offsets in the Nodes array

            // Offset 12
            nodeFIELD,        // canField.nodeClass
            canFIELD,         // canField.canOp
            2,                  // canField.iFieldNum
            0,                  // canField.iNameOffset: szField
                                //     is the literal at offset 0

            // Offset 20
            nodeCONST,        // canConst.nodeClass
            canCONST,         // canConst.canOp
            fldZSTRING,       // canConst.iType
            6,                  // canConst.iSize
            5                   // canConst.iOffset: szConst is
                                //     the literal at offset
                                //     strlen(szField) + 1
        };

// Name of the field for the third node of the tree.
CHAR                szField[] = "NAME";
// Value of the constant for the second node of the tree.
CHAR                szConst[] = "OCEAN";
// Return value from IDAPI functions.
DBIResult           rslt;

Screen("*** Filter Example ***\r\n");

BREAK_IN_DEBUGGER();

Screen("      Initializing IDAPI...");
if (InitAndConnect(&hDb) != DBIERR_NONE)
{
    Screen("\r\n*** End of Example ***");
    return;
}

Screen("      Setting the database directory...");
rslt = DbiSetDirectory(hDb, (pCHAR) szTblDirectory);
ChkRslt(rslt, "SetDirectory");

Screen("      Open the %s table...", szTblName);
```

```
rslt = DbiOpenTable(hDb, (pCHAR) szTblName,
                         (pCHAR) szTblType, NULL, NULL, 0,
                         dbiREADWRITE, dbiOPENSHARED, xltFIELD,
                         FALSE, NULL, &hCur);
if (ChkRslt(rslt, "OpenTable") != DBIERR_NONE)
{
     CloseDbAndExit(&hDb);
     Screen("\r\n*** End of Example ***");
     return;
}

// Go to the beginning of the table
rslt = DbiSetToBegin(hCur);
ChkRslt(rslt, "SetToBegin");

Screen("\r\n      Display the %s table...", szTblName);
DisplayTable(hCur, uNumRecs);

uSizeNodes         = sizeof(Nodes);     // Size of the nodes.
// Size of the literals.
uSizeLiterals     = strlen(szField) + 1 + strlen(szConst) + 1;
// Size of the header information.
uSizeCanExpr      = sizeof(CANExpr);
// Total size of the filter.
uTotalSize         = uSizeCanExpr + uSizeNodes + uSizeLiterals;

// Initialize the header information
canExp.iVer = 1;                          // Version.
canExp.iTotalSize = uTotalSize;      // Total size of the
                                          //     filter.
canExp.iNodes = 3;                        // Number of nodes.
canExp.iNodeStart = uSizeCanExpr;   // The offset in the
                                          //     buffer where the
                                          //     expression nodes
                                          //     start.

// The offset in the buffer where the literals start.
canExp.iLiteralStart = uSizeCanExpr + uSizeNodes;
// Allocate space for the filter expression.
pcanExpr = (pBYTE)malloc(uTotalSize * sizeof(BYTE));
if (pcanExpr == NULL)
{
     Screen("      Could not allocate memory...");
     DbiCloseCursor(&hCur);
     CloseDbAndExit(&hDb);
     Screen("\r\n*** End of Example ***");
     return;
}

// Initialize the filter expression.
memmove(pcanExpr, &canExp, uSizeCanExpr);
memmove(&pcanExpr[uSizeCanExpr], Nodes, uSizeNodes);

memmove(&pcanExpr[uSizeCanExpr + uSizeNodes],
          szField, strlen(szField) + 1); // First litteral
memmove(&pcanExpr[uSizeCanExpr + uSizeNodes +
```

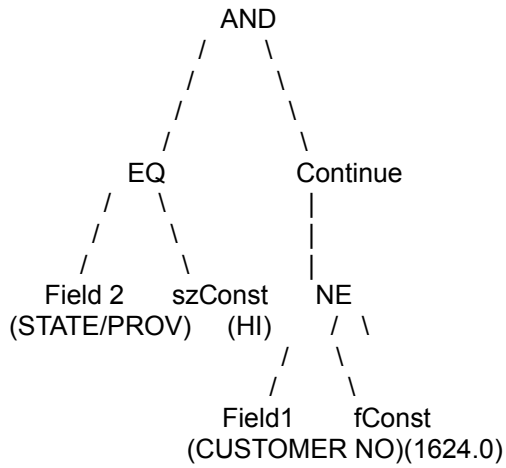
```
                        strlen(szField) + 1],
                szConst, strlen(szConst) + 1); // Second litteral

    rslt = DbiSetToBegin(hCur);
    ChkRslt(rslt, "SetToBegin");

    Screen("\r\n      Add a filter to the %s table which will"
           " limit the records\r\n            which are displayed"
           " to those whose %s field starts with '%s'...",
           szTblName, szField, szConst);
    rslt = DbiAddFilter(hCur, 0L, 1, FALSE, (pCANExpr)pcanExpr,
                        NULL, &hFilter);
    if (ChkRslt(rslt, "AddFilter") != DBIERR_NONE)
    {
        rslt = DbiCloseCursor(&hCur);
        ChkRslt(rslt, "CloseCursor");
        free(pcanExpr);
        CloseDbAndExit(&hDb);
        Screen("\r\n*** End of Example ***");
        return;
    }

    // Activate the filter.
    Screen("      Activate the filter on the %s table...",
           szTblName);
    rslt = DbiActivateFilter(hCur, hFilter);
    ChkRslt(rslt, "ActivateFilter");

    rslt = DbiSetToBegin(hCur);
    ChkRslt(rslt, "SetToBegin");

    Screen("\r\n      Display the %s table with the filter"
           " set...", szTblName);
    DisplayTable(hCur, uNumRecs);

    Screen("\r\n      Deactivate the filter...");
    rslt = DbiDeactivateFilter(hCur, hFilter);
    ChkRslt(rslt, "DeactivateFilter");

    Screen("\r\n      Drop the filter...");
    rslt = DbiDropFilter(hCur, hFilter);
    ChkRslt(rslt, "DropFilter");

    rslt = DbiCloseCursor(&hCur);
    ChkRslt(rslt, "CloseCursor");

    free(pcanExpr);

    Screen("      Close the database and exit IDAPI...");
    CloseDbAndExit(&hDb);

    Screen("\r\n*** End of Example ***");
}
```

subject to the terms of the No-Nonsense License Statement that you received with the Borland product to which this information pertains.

Using a continue node in a filter.

```
   NUMBER   :   2763
  PRODUCT   :   BDE
  VERSION   :   All
       OS   :   Windows
     DATE   :   August 5, 1996

    TITLE   :   Using a continue node in a filter.
```

```
/*

    This example shows the use of a canContinue node. A
    Continue node is used to stop evaluating when a certain
    condition is false for the first time.

    This filter will limit the result set to those customers
    living in Hawaii, you are listed in the table before the
    customer with ID 1624.

    Picture of the filter expression:
```




```
*/

// BDE - (C) Copyright 1994 by Borland International

// Filter.c
#include "snipit.h"

static const char szTblName[] = "customer";
static const char szTblType[] = szPARADOX;

//==================================================================
//   Function:
//              Filter();
//
//   Description:
```

```
//              This example shows how to use filters to limit the
//              result set of a table. Filters perform a function
//              similar to that of ranges, but more powerful
//              operations are supported.
//============================================================
void
Filter (void)
{
    hDBIDb          hDb;            // Handle to the database.
    hDBICur         hCur;           // Handle to the table.
    pBYTE           pcanExpr;       // Structure containing
                                    //    filter info.
    hDBIFilter      hFilter;        // Filter handle.
    UINT16          uSizeNodes;     // Size of the nodes in the
                                    //    tree.
    UINT16          uSizeCanExpr;   // Size of the header
                                    //    information.
    UINT16          uSizeLiterals;  // Size of the literals.
    UINT16          uTotalSize;     // Total size of the filter
                                    //    expression.
    UINT32          uNumRecs   = 10; // Number of records to
                                    //    display.
    CANExpr         canExp;         // Contains the header
                                    //    information.

    UINT16          Nodes[] =       // Nodes of the filter tree.
        {
            // Offset 0. Node 1.
            nodeBINARY,     // canBinary.nodeClass
            canAND,         // canBinary.canOp
            8,              // canBinary.iOperand1 - node 2
            34,             // canBinary.iOperand2 - node 5
                            //    Offsets in the Nodes array

            // Offset 8. Node 2.
            nodeBINARY,     // canBinary.nodeClass
            canEQ ,         // canBinary.canOp
            16,             // canBinary.iOperand1 - node 3
            24,             // canBinary.iOperand2 - node 4
                            //    Offsets in the Nodes array

            // Offset 16. Node 3.
            nodeFIELD,      // canField.nodeClass
            canFIELD,       // canField.canOp
            5,              // canField.iFieldNum
            11,             // canField.iNameOffset: szField2
                            //    is the literal at offset
                            //    strlen(szField1) + 1

            // Offset 24. Node 4.
            nodeCONST,      // canConst.nodeClass
            canCONST,       // canConst.canOp
            fldZSTRING,     // canConst.iType
            3,              // canConst.iSize
            31,             // canConst.iOffset: fconst is
                            //    the literal at offset
```

```
                                //    strlen(szField1) + 1 +
                                //    sizeof(fConst) +
                                //    strlen(szField2) + 1

                // Offset 34. Node 5.
                nodeUNARY,          // canBinary.nodeClass
                canCONTINUE,        // canBinary.canOp
                40,                 // canBinary.iOperand1 - node 6
                                //    Offsets in the Nodes array

                // Offset 40. Node 6.
                nodeBINARY,         // canBinary.nodeClass
                canNE ,              // canBinary.canOp
                48,                 // canBinary.iOperand1 - node 7
                56,                 // canBinary.iOperand2 - node 8
                                //    Offsets in the Nodes array

                // Offset 48. Node 7.
                nodeFIELD,          // canField.nodeClass
                canFIELD,           // canField.canOp
                1,                  // canField.iFieldNum
                0,                  // canField.iNameOffset:
                                //    szField1 is the literal at
                                //    offset 0.

                // Offset 56. Node 1.
                nodeCONST,          // canConst.nodeClass
                canCONST,           // canConst.canOp
                fldFLOAT,           // canConst.iType
                8,                  // canConst.iSize
                23,                 // canConst.iOffset: fconst is
                                //    the literal at offset
                                //    strlen(szField1) + 1 +
                                //    strlen(szField2) + 1
        };

// Name of the field for the third node of the tree.
CHAR                szField1[] = "CUSTOMER NO";
// Name of the field for the third node of the tree.
CHAR                szField2[] = "STATE/PROV";
// Value of the constant for the second node of the tree.
FLOAT               fConst        = 1624.0;
// Value of the constant for the second node of the tree.
// Field #7
CHAR                szConst[]    = "HI";

DBIResult           rslt; // Return value from IDAPI functions.

Screen("*** Filter Example ***\r\n");

BREAK_IN_DEBUGGER();

Screen("      Initializing IDAPI...");
if (InitAndConnect(&hDb) != DBIERR_NONE)
{
    Screen("\r\n*** End of Example ***");
```

```
    return;
}

Screen("    Setting the database directory...");
rslt = DbiSetDirectory(hDb, (pCHAR) szTblDirectory);
ChkRslt(rslt, "SetDirectory");

Screen("    Open the %s table...", szTblName);
rslt = DbiOpenTable(hDb, (pCHAR) szTblName,
                    (pCHAR) szTblType,
                    NULL, NULL, 0, dbiREADWRITE,
                    dbiOPENSHARED, xltFIELD, FALSE, NULL,
                    &hCur);
if (ChkRslt(rslt, "OpenTable") != DBIERR_NONE)
{
    CloseDbAndExit(&hDb);
    Screen("\r\n*** End of Example ***");
    return;
}

// Go to the beginning of the table
rslt = DbiSetToBegin(hCur);
ChkRslt(rslt, "SetToBegin");

Screen("\r\n    Display the %s table...", szTblName);
DisplayTable(hCur, uNumRecs);

// Size of the nodes.
uSizeNodes      = sizeof(Nodes);
// Size of the literals.
uSizeLiterals   = strlen(szField1) + 1 + sizeof(fConst) +
                  strlen(szField2) + 1 + strlen(szConst)
                  + 1;
// Size of the header information.
uSizeCanExpr    = sizeof(CANExpr);
// Total size of the filter.
uTotalSize      = uSizeCanExpr + uSizeNodes + uSizeLiterals;
// Initialize the header information
canExp.iVer = 1; // Version.
canExp.iTotalSize = uTotalSize; // Total size of the filter.
canExp.iNodes = 8; // Number of nodes.
canExp.iNodeStart = uSizeCanExpr; // The offset in the
                                  //   buffer where the
                                  //   expression nodes
                                  //   start.
// The offset in the buffer where the literals start.
canExp.iLiteralStart = uSizeCanExpr + uSizeNodes;

// Allocate space for the filter expression.
pcanExpr = (pBYTE)malloc(uTotalSize * sizeof(BYTE));
if (pcanExpr == NULL)
{
    Screen("    Could not allocate memory...");
    DbiCloseCursor(&hCur);
    CloseDbAndExit(&hDb);
    Screen("\r\n*** End of Example ***");
```

```
    return;
}

// Initialize the filter expression.
memmove(pcanExpr, &canExp, uSizeCanExpr);
memmove(&pcanExpr[uSizeCanExpr], Nodes, uSizeNodes);

memmove(&pcanExpr[uSizeCanExpr + uSizeNodes],
        szField1, strlen(szField1) + 1); // First litteral

memmove(&pcanExpr[(uSizeCanExpr + uSizeNodes +
                   strlen(szField1) + 1)],
        szField2, strlen(szField2) + 1); // First litteral


memmove(&pcanExpr[(uSizeCanExpr + uSizeNodes +
                   strlen(szField1) + 1 +
                   strlen(szField2) + 1)],
        &fConst, sizeof(fConst)); // Second litteral

memmove(&pcanExpr[(uSizeCanExpr + uSizeNodes +
                   strlen(szField1) + 1 +
                   strlen(szField2) + 1 + sizeof(fConst))],
        szConst, strlen(szConst) + 1); // First litteral

rslt = DbiSetToBegin(hCur);
ChkRslt(rslt, "SetToBegin");

Screen("\r\n      Add a bug filter to the %s table which will"
       " limit the records\r\n            which are displayed"
       " to those whose %s field is greater than %.1lf...",
       szTblName, szField1, fConst);
rslt = DbiAddFilter(hCur, 0L, 1, FALSE, (pCANExpr)pcanExpr,
                    NULL, &hFilter);
if (ChkRslt(rslt, "AddFilter") != DBIERR_NONE)
{
    rslt = DbiCloseCursor(&hCur);
    ChkRslt(rslt, "CloseCursor");
    free(pcanExpr);
    CloseDbAndExit(&hDb);
    Screen("\r\n*** End of Example ***");
    return;
}

// Activate the filter.
Screen("      Activate the filter on the %s table...",
       szTblName);
rslt = DbiActivateFilter(hCur, hFilter);
ChkRslt(rslt, "ActivateFilter");

rslt = DbiSetToBegin(hCur);
ChkRslt(rslt, "SetToBegin");

Screen("\r\n      Display the %s table with the filter"
       " set...", szTblName);
DisplayTable(hCur, uNumRecs);
```

```
    Screen("\r\n      Deactivate the filter...");
    rslt = DbiDeactivateFilter(hCur, hFilter);
    ChkRslt(rslt, "DeactivateFilter");

    Screen("\r\n      Drop the filter...");
    rslt = DbiDropFilter(hCur, hFilter);
    ChkRslt(rslt, "DropFilter");

    rslt = DbiCloseCursor(&hCur);
    ChkRslt(rslt, "CloseCursor");

    free(pcanExpr);

    Screen("      Close the database and exit IDAPI...");
    CloseDbAndExit(&hDb);

    Screen("\r\n*** End of Example ***");
}
```

Delphi Fact Sheet

NUMBER : 2767
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Delphi Fact Sheet

Delphi Fact Sheet

Delphi is the only Windows development tool to combine the Rapid Application Development (RAD) benefits of visual component-based design with the power of an optimizing native code compiler and scalable database access.

Delphi represents the next generation in:

- Performance--with the world's fastest compiler
- Rapid Application Development--via visual Two-Way-Tools()
- Component Reuse--a true object-oriented environment
- Scalable Database Access--the fast track to client/server

The fastest way to the fastest applications

Delphi is the next generation Windows development tool, combining the most intuitive visual design environment with the unrivaled performance of a world-class optimizing native code compiler. It compiles Windows applications at more than 350,000 lines per minute.* An automatic MAKE facility ensures that only code that has been changed is recompiled for maximum efficiency, and the built-in assembler allows for ultimate fine-tuned performance.

The resulting executable files (EXEs) are lean, highly efficient, and run up to 10 to 20 times faster than interpreted p-code. Delphi executables are immediately deployable, royalty-free, and require no runtime interpreter Dynamic Link Library (DLL), so maximum performance is guaranteed.

RAD, from prototype to production

The comprehensive suite of visual design and debugging tools accelerate development so you move seamlessly from prototyping to deployment. You'll create high-performance applications in record time with the gallery of reusable forms, project templates, programming experts, and context-sensitive intelligent help.

And smart computer-based tutorials guide you through your project and help explain concepts. The Object Inspector; provides complete access to properties and events of a selected component, with smart Property Editors for rapid

design changes. The Project Manager provides an overview of all forms and code for a development project, while the ObjectBrowser() provides a graphical view of the overall object hierarchy.

The sophisticated GUI Debugger jumps directly to the point in the source code where problems occur, so you can find and fix problems quickly and easily. And full support for conditional breakpoints, watchpoints, call stack monitoring, single-step execution, and trace mode are included.

A suite of powerful Windows resource editors lets you create and modify menus, icons, cursors, and bitmaps for low-level debugging. The WinSight() utility allows you to monitor the Windows messaging flow, while WinSpector() instantly provides comprehensive diagnostic details.

Visual Two-Way-Tools for power programming

With Delphi, you can always get to your code, and everything you can do visually can also be done in code. Innovative Two-Way-Tools maximize your productivity by letting you move seamlessly between the visual design environment and the synchronized underlying source code.

Switching between the visual design mode and the intelligent BRIEF;-style source code editor is rapid and intuitive. It supports advanced editing features such as Color Syntax Highlighting, macro recording, column block marking, and regular expression support.

Reuse components to maximize productivity

The comprehensive Visual Component Library (VCL), a collection of more than 75 objects such as dialogs, buttons, and list boxes, includes a host of additional reusable objects including database controls, notebook tabs, grids, multimedia controls, and much more!

The completely customizable Integrated Development Environment can be tailored to individual or corporate preferences. Custom reusable components and commercial VBXs are easily added to the fully reconfigurable Component Palette. You can create and install your own application and form templates. And the Delphi component-based architecture allows you to seamlessly integrate DLLs, VBBX controls, and OLE 2.0 servers into your applications.

Integrated database support and reporting capabilities

Delphi incorporates the Borland Database Engine (BDE), providing direct access to data stored in dBASE, Paradox, and the Local InterBase Server, and to other data formats via ODBC. And data-aware controls allow manipulation of live data at design time, improving project turnaround time.

A comprehensive selection of Visual Data Objects helps make sophisticated database application design a breeze. And the included Database Desktop() utility lets you create or restructure tables and manage connections, insulating compiled Delphi applications from changes in database locations.

Delphi also includes the PC version of ReportSmith; Borland's award-winning database reporting and query tool that supports live data access at design time. ReportSmith supports a wide range of popular database formats, and

provides a variety of graphing, tabulating, and charting options.

The ultimate complementary tool

If you prefer to work with a variety of programming tools, Delphi is the perfect addition to your toolset. It allows you to create reusable DLLs that can be called from applications built with C++, Paradox, dBASE, PowerBuilder, Visual Basic, and other popular development tools. Delphi can also use standard VBX controls, OLE2.0 servers, DLLs created by other development tools, Microsoft multimedia, MAPI, and pen computing APIs. No matter what you program with today, the Delphi component-based design makes it a valuable addition that leverages your existing investment in code.

Delphi gives you the control and flexibility you demand with complete access to the Windows API, while shielding you from the complexities of Windows programming.

Object-Oriented Programming

The underlying Object Pascal language has been enhanced to fully support exception handling. All your GPFs, disk I/O errors, and other dreaded problems are automatically trapped. And it offers all of the benefits of a structured programming language and a true object-oriented development tool, including polymorphism, inheritance, and encapsulation. So Delphi applications are incredibly robust and deliver superior solutions for all of your mission-critical projects.

Enhanced compiler features such as conditional compilation and smart linking make it even easier to rapidly design professional applications intended for a variety of target environments.

Fast track to Windows 95

Delphi applications developed for the Windows 3.1 operating environment will also run under Windows 95 and Windows NT. Upgrading them to full 32-bit performance simply involves a

single-click recompile with the forthcoming Windows 95 version of Delphi. No rewriting of code is necessary!

Smooth scaling to client/server

Delphi provides an easy path to the fast-growing market of client/server applications development. Transparent local SQL development is possible with Delphi, using the built-in Local InterBase Server, which enables you to rapidly develop high-performance ANSI SQL-92 compliant applications for standalone systems.

To make the transition from Delphi to professional client/server development and deployment using external database hosts, upgrade to Delphi Client/Server. It's a quick and easy process requiring no additional coding. Delphi Client/Server features high-performance native drivers for Oracle, Sybase, Informix, and InterBase, along with team development support, a Visual Query Builder tool, and much more.

Delphi, the developer's solution

- Quickly and easily customize the Component Palette.
- Delphi gives you everything you need for Rapid Application Development.
- Manage all your code and visual objects with the Object Inspector.
- Two-Way-Tools give you complete control of your code.
- The Visual Component Library lets you build complete database applications in minutes.

Quotes:

"It's going to change our lives, you know."
--J.D. Hildebrand, Editor
Windows Tech Journal

"When it comes to keeping required coding to a minimum, Delphi excels..."
--Windows Sources
December 1994

"Delphi gives developers exactly what they need to create complete applications, posthaste."
--Susan Ryan
InfoWorld

"5 Stars--Excellent"
--PC/Computing
February 1995

Delphi Specifications

Optimizing Native Code Compiler

- Compiles at over 350,000 lines per minute*

- Create fast standalone EXEs with no runtime interpreter Dynamic Link Library (DLL)
- Applications run up to 10 to 20 times faster than interpreted p-code
- Create DLLs that work with C++, dBASE, Paradox, Visual Basic, PowerBuilder, and others
- Access to all Windows API functions and messages
- Optimized case statements, sets, 32-bit math operations, string and file routines, and more
- Math coprocessor and emulator support
- Automatic built-in MAKE facility
- Conditional compilation
- Smart Linker removes unused objects and code
- Command-line compiler and MAKE facility
- Built-in assembler for tuning performance
- Linker optimization for smaller EXEs

Integrated Development Environment

- Integrated visual form designer with over 75 components
- Two-Way-Tools automatically synchronize code and visual representations
- Object Inspector allows visual customization of components without writing code
- Built-in tools for alignment, scaling, sizing, and tab order
- ObjectBrowser displays object hierarchy, units, globals, and code references
- Project Manager displays all files and forms
- Customizable environment including SpeedBar, Component Palette, editor, and browser
- Intuitive Icon, Bitmap, and Menu editors
- Open environment for adding your own tools, experts, components, and Property Editors

Visual Component Library (VCL)

- Customizable palette of over 75 reusable componnents
- Support for the latest Microsoft systems technologies including OLE 2.0, DDE, VBXs, DLLs, MAPI, and ODBC
- Standard components for menus, bitmapped buttons, masked edit fields, panels, graphics, notebook tabs, grids, outlines, list boxes, combo boxes, check boxes, labels, and more
- Visual Data Objects for accessing databases, tables, queries, reports, SQL stored procedures, as well as data-aware grids, navigators, lookup lists, edit fields, list boxes, combo boxes, memos, bitmaps, and more
- "Live" design-time data access
- Extend the VCL at any time with third-party libraries or with your own objects created with Delphi
- Database Field Editor for validation rules, display format, edit mask, and field width

- Standard dialog objects for file operations, printing, font and color selection, and searching
- System objects for accessing OLE 2.0 servers, DDE, multimedia and file, and directory lists

- Support for MDI, printing, annd graphics
- 2-D and 3-D charts with included ChartFX control
- Compatible with hundreds of VBX controls
- Includes sample objects with complete source code written in Delphi
- VCL source code available separately

Object Pascal Language

- High-performance, structured, object-oriented language
- Complete support for inheritance, polymorphism, and encapsulation
- Control over privacy with Public, Private, Protected, and Published reserved words
- Create components with properties and events
- Use inheritance to customize any object
- Automatic runtime type information and object persistence
- Automatic, extensible exception handling
- Support for open arrays, user-defined types, objects, and pointers
- Advanced language support for delegation and class references
- Separate compilation of units and DLLs
- Over 150 library routines for mathematical operations, string manipulation, text formatting, and file management
- WinCRT unit for ccreating standard Pascal "console" programs
- Compiles Borland Pascal 7.0 for Windows code

Powerful Editing

- Full-featured BRIEF-style editing
- Unlimited undo and redo
- No limit on file size
- Macro record and playback
- Column block marking
- Regular expression (GREP-style) search
- Customizable Color Syntax Highlighting
- Selectable editor keystrokes with support for BRIEF and Epsilon

Integrated Debugging

- Integrated GUI Debugger with single-step and trace
- Conditional breakpoints and watchpoints
- Call stack view for tracing code execution
- Evaluate and modify expressions and variables
- WinSight Windows message trace utility
- WinSpector postmortem analysis tool
- Compatible with Turbo Debugger for Windows (available separately)

Borland Database Engine

- High-performance engine with native drivers for dBASE, Paradox, and Local InterBase Server
- Fully scalable support for migrating applications from desktop to client/sserver
- Royalty-free deployment of database engine
- Space-efficient deployment of database applications

- Includes Database Desktop for managing database aliases as well as creating and restructuring tables
- ODBC support for Access, Btrieve, Excel, DB2, AS/400, Ingres, HP ALBASE/SQL, and gateways like IBM DDCS/2, Micro Decisionware and Sybase Net-gateway. (available separately)
- Delphi Client/Server includes SQL Link native drivers for InterBase, Oracle, Sybase, Microsoft SQL Server, and Informix. (available separately)

Local InterBase 4.0 Server

- High-performance ANSI SQL-92 compliant
- Ideal for "off-line" development and single-user applications
- Computed fields
- Outer joins and join expressions in the where clause
- Complex data including Binary Large Objects (BLObs()) and multidimensional arrays
- Advanced features including stored procedures, triggers, and constraints
- Fully scalable to InterBase on NT, NetWare, and UNIX platforms
- Local InterBase Server Deployment Kit available separately

ReportSmith 2.5

- Award-winning database reporting and query tool
- Access ReportSmith from the Delphi Tools menu and TReport component
- Features "live" report writing
- Adaptive data access works with any size database
- Prebuilt report templates and styles
- Crosstab and mailing label reports
- Multilevel sorting
- Custom group specifications
- Onscreen Print Preview mode
- Free distribution of ReportSmith runtime module
- Fully compatible with ReportSmith SQL edition

Documentation and Help

- Extensible gallery of application templates and experts
- Seven Interactive Tutors() with "live" interaction
- Over 5Mb of on-line help
- Five manuals with over 1,200 pages of documentation: Delphi User's Guide, Component Writer's Guide, Database Application Developer's Guide, Creating Reports, and InterBase User's Guide

Minimum System Requirements

- Intel 386-baased PC or higher
- Microsoft Windows 3.1 or later, 100%-compatible version
- 6Mb of extended memory or higher
- 30Mb hard disk space
- CD-ROM drive (3.5" disks available separately)

Networks Supported

Any Microsoft Windows 3.1 compatible network, including OS/2 2.1, Novell NetWare, Windows for Workgroups 3.11, and Windows NT 3.11.

*Pentium 90MHz

Corporate Headquarters: 100 Borland Way, Scotts Valley, California 95066-3249, 408-431-1000. Offices in: Australia (61-02-911-1000), Belgium (32-2-47-99-903), Brazil (55-11-851-5326), Canada (416-229-6000), Chile (56-2-233-7113), Denmark (45-48-14-00-01), France (33-1-41-23-11-00), Germany (49-6103-9790), Hong Kong (852-2540-4380), Italy (39-2-26-91-51), Japan (81-03-5350-9370), Korea (82-02-551-2795), Latin American Headquarters in U.S.A. (408-431-1074), Malaysia (60-03-230-2618), Netherlands (31-020-540-54-00), New Zealand (64-9-443-8890), Singapore (65-339-8122), Spain (34-1-650-72-50), Sweden (46-8-634-35-00), Switzerland (41-031-761-2604) Taiwan (886-2-718-6627), and United Kingdom (44-734-321-150)

List of Delphi Books From Third-Party Publishers

NUMBER : 2776
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : List of Delphi Books From Third-Party Publishers

The Delphi Programmer Explorer
by J. Duntemann/J. Mischel/D. Taylor
Coriolis Group
ISBN: 1-883577-25-X $39.99
A new type of tutorial: Theory and practice alternate in short chapters, with the emphasis on creating useful software starting on the very first page.

Delphi for Dummies
by Neil Rubenking
IDG Press
ISBN: 1-56884-200-7 $19.99
Readers will learn about Borland's new language in the easy to understand style of the Dummies series.

Teach Yourself Delphi
by Devra Hall
MIS Press
ISBN: 1-55828-390-0 $27.95
Here is a complete, self-guided tour to the new development environment from Borland, encompassing all the features of the language and all the tools, tricks, and advantages of Delphi.

Delphi Nuts and Bolts
by Gary Cornell and Troy Strain
Osborne-McGraw-Hill
ISBN: 0-07-882-136-3 $24.95
If you are an experienced programmer and want a fast introduction to Delphi, this book is for you.

Software Engineering with Delphi
by Edward C. Webber, J. Neal Ford, and Christopher R. Webber
Prentice Hall Professional, Trade & Reference
A guide to developing client/server applications with an emphasis on Delphi's object-oriented tools.

Delphi by Example
by Blake Watson
Que
ISBN: 1-56529-757-1 $29.99
Aimed at the beginning programmer who has no prior experience with other languages or development products, this book presents basic concepts of programming along with a clear explanation of

the key development tools that are part of Delphi.

Using Delphi, Special Edition
by John Matcho and Eric Uber
Que
ISBN: 1-56529-823-3 $29.99
This 3-part tutorial on the most important Delphi features covers how to install the product and develop applications using Delphi's visual tools, explores the Windows application development process, and deals with some advanced programming topics.

Developing Client/Server Applications with Delphi
by Vince Killen and Bill Todd
Sams Publishing
Walks the reader through the development process of creating real-world C/S applications, explaining in detail what the thought processes must be even before any code is written.

Delphi Developer's Guide
by Xavier Pacheco/Steve Teixeira
Sams Publishing
ISBN: 0-672-30704-9 $45.00
Intermediate to advanced guide to developing applications using Delphi.

Master Delphi
by Charlie Calvert
Sams Publishing
ISBN: 0-672-30499-6 $45.00
Comprehensive tutorial/reference for intermediate programming with Delphi.

Teach Yourself Delphi in 21 Days
by Andrew Wozniewicz
Sams Publishing
ISBN: 0-672-30470-8 $29.99
Introduces Delphi to the beginning programmer and includes question-and-answer section at end of each less to test readers progress as they learn.

Mastering Delphi
by Marco Cantu
Sybex
ISBN: 0-7821-1739-2 $29.99
Introduces programmers to Delphi's features and techniques, including secrets of the environment, the programming language, the custom components and Windows programming in general.

Delphi How-To
by Gary Frerking

Waite Group Press
ISBN: 1-57169-019-0 $34.95
Presents large collection of programming problems and their solutions in standard, easy-to-use reference format, including unique solutions that use VBX controls and easy ways to build multimedia projects with Delphi.

Developing Windows Applications Using Delphi
by Paul Penrod
John Wiley
ISBN: 0-471-11017-5 $29.95
This introduction for traditional C programmers who want to make the transition to rapid application development also provides detailed instructions for building sophisticated Windows applications and for creating graphical interfaces.

Instant Delphi
by Dave Jewell
Wrox Press
ISBN: 1-874416-57-5 $19.95
Instant Delphi is the fast-paced tutorial guide for the programmer who wants to get up to speed on the Delphi product as quickly as possible.

BDE Frequently Asked Questions.

NUMBER : 2770
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : January 23, 1997

TITLE : BDE Frequently Asked Questions.

This document contains information most often provided to users of this section. First, there is a listing of common Technical Information Documents (TI's) that can be downloaded from the Borland FTP site, CompuServe, and the Borland Download BBS. The documents are also available through the TechFax line of Borland Assist. Following the TI list, there is a listing of the most frequently asked questions and answers.

TECHNICAL INFORMATION DOCUMENTS

TI2546.ZIP Problem Report Form
TI2656.ZIP Function mapping from the Paradox Engine to BDE
TI2751.ZIP Some Internal BDE Limits (can change at any time)
TI2752.ZIP Local SQL Reserved words
TI2761.ZIP Getting started using the BDE
TI2762.ZIP Comparison filters
TI2763.ZIP Using a continue node in a Filter
TI2817.ZIP Understanding the PARADOX.NET file with the BDE
TI2822.ZIP Setting File Handles For A Windows BDE Application
TI2919.ZIP Changing the NET DIR Programmatically
TI2989.ZIP BDE setup for Peer-To-Peer(Non-Dedicated) Networks
TI2993.ZIP Removing "Lock file has grown too large" Error
TI3089.ZIP Sharing Violation Error with Paradox Tables
TI3188.ZIP Steps for FAT32 Support with the BDE
TI9410.ZIP Overview information on the BDE

FREQUENTLY ASKED QUESTIONS AND ANSWERS
---

1) General
2) Paradox Table Specifics
3) dBASE Table Specifics

GENERAL

Q: Is there any "Getting started" information on the BDE?

A: Yes - get the file TI2761.ZIP for information on getting started with the BDE.

Q: What's the difference between Paradox for Windows and the Borland Database Engine (BDE)?

A: Paradox for Windows is a database Application and development system. The BDE is a programming tool allowing C, C++, and Pascal programmers access to Paradox, dBASE, Text, as well as other data sources using either the Borland SQL Links for Windows native drivers, or using third-party ODBC drivers. The BDE contains the core DLLs that Paradox for Windows, as well as dBASE for Windows, use for their core data access. More general information on the BDE can be found in the file TI9410.ZIP.

Q: What basic steps can I follow to make my BDE application run smoothly?

A: 1) Increase stack size to 20K. (16 bit only)
2) Increase the number of file handles available to your application using the Windows API function SetHandleCount. (16 bit only)
3) Check the return values of each and every BDE function call and provide some means for handling the result of any return value other than DBIERR_NONE. (See next question also).
4) Take a look at TI2761.ZIP, which describes the basic steps that are required in setting up a BDE application.

Q: I'm having trouble with my program, and the debugger has traced the problem into a BDE DLL. Does this mean that there is a bug in the BDE?

A: Not necessarily. Frequently if a prior call to the BDE has failed due to being passed an invalid handle or for some other reason, this will leave the BDE in an unpredictable state which will later cause a GP fault. The solution is to check the return values of each and every function call you make. The BDE sample applications contain error handling routines.

Q: Why is my application having problems when share is loaded?

A: One possibility is that share could be running out of locks or file handles. The command line 'share /L:200 /F:4096' will increase the number of files that can be locked to 200 and the memory available for files to 4096 (the default is 20 and 2048). See your DOS or WINDOWS manual for more information on share.

This information only applies to Windows 3.1. Windows for Workgroups, Windows 95 and Windows NT load share automatically. Windows 3.1 users can get this same functionality and not have to worry about running out of locks or file handles by obtaining the Windows 3.1 version of VSHARE.DLL from Microsoft.

Q: Is there a version of the Database Framework (DBF) for the BDE? Is there a C++ Framework for the BDE?

A: An example framework, roughly 95% compatible with the Paradox Engine DBF, is available on CompuServe in the BDEVTOOLs forum, LIBrary 4: KDBF.ZIP.

Q: Why am I getting the error: "Could not initialize IDAPI - not initialized for accessing network drives" when attempting to open a table on a network?

A: 1) Make certain the network control directory (NET DIR) is set to a directory on the network (not local). This is set in the BDE Configuration utility.
2) Make certain that you have read/write/create rights to the network control directory, as well as the directory containing the table.
3) Make certain that the private directory is set to a local directory. For example, in the Database Desktop (DBD), the private directory is set in the WIN.INI file, in the [DBD] section, PrivDir. If it is set to a directory on the network, this error can occur.
4) Old lock files exist in the private directory. Either the application previously crashed, resulting in improper cleanup, or another application is using the same private directory.
5) Under Windows for Workgroups (WfW), search your hard drive for the NWCALLS.DLL. Rename/backup the older versions of this file and run again.

If this fails to resolve the problem, you will need to make certain to not use the NetBEUI protocol as the default network protocol in Windows Setup. Select NetWare (or some other protocol) as the networking protocol, and everything should work successfully.

Q: Why am I getting the error "Cannot find NetWare.DLL"?

A: An outdated netware.drv is found in the System.ini file. A newer version of this file should be gotten from your network administrator, and/or Novell.

Q: How do I use the BDE in a DLL that is called from Paradox for Windows or dBASE for Windows?

A: Make certain to use dynamic linking when using the BDE in a DLL called from another application. This will be done automatically when using the IDAPI.LIB that ships with the BDE, but will not be done if an import library was created using IMPLIB (the shipping IDAPI.LIB is more than a standard import library).

Q: Why am I having problems getting information from the DbiGetFieldDescs function?

A: Make certain to have 'Allocate Enums as Ints' set (Options | Compiler | Code Generation in BC 4.x, Options | Project | Compiler | Code Generation in BC 5.). The FLDDesc structure, which is used by DbiGetFieldDesc, contains an enum, which within the DLL is set to the same size as an int (two bytes).

Q: Why am I having stack corruption problems? or Why is my application crashing when calling or returning from a function?

A: First, make certain that you have 25k of stack allocated for your application. Then, make certain that you have 'Allocate enums as ints' selected. A number of structures, including CURProps, make use of enumerations. As there enumerations are allocated two bytes within the DLL, we need to make certain that the application is passing two bytes as well. This is done with the 'Allocate enums as ints' option.

Q: How can I optimize BDE performance on table operations?

A: Although there are a number of ways to improve BDE performance, some general things to try are as follows:

1) Keep the number of maintained secondary indexes to a minimum; sometimes it is better to delete the index and recreate it than to perform a number of table operations with the indexes in place.
2) If possible, increase the size of the swap buffer and the number of file handles that the BDE has available to it. This will decrease the Engine's need to swap resources. Note: make certain to increase the file handles available to your application using SetHandleCount, as well as increasing the number of file handles available to the BDE in IDAPI.CFG.
3) Open the table exclusively.
4) Batch as many opperations as possible - do not read/ write records one as a time. Use DbiBatchMove, DbiCopyTable, DbiReadBlock, and/or DbiWriteBlock.
5) When using DbiWriteBlock, try to work in multiples of the physical block size, usually 2k or 4k.
6) If you are opening and closing one or more tables repeatedly, consider calling DbiAcqPersistTableLock on a non-existent file after you initialize the BDE. This will create the .LCK file so that it will not have to be created each time a table is opened, created, etc. (Note: you'll also want to call DbiRelPersistTableLock before calling DbiExit).
7) Used Cached Updates
   Paradox Tables Only.
8) Work with in-memory tables when possible (Note that in-memory tables cannot be the source table to DbiBatchMove).

Q: What are the current versions of the BDE?

A: For 16 bit, it is 2.52. It may be obtained by downloading BDE252.ZIP.
For 32 bit, it is 3.5. It may be obtained by downloading BDE35F.ZIP.

Q: Can I use the DOS Power Pack with the Borland Database Engine?

A: No. The BDE makes use of Windows API functions that are not emulated by the DOS Power Pack.

Q: Why am I getting the error 'Invalid BLOB Handle in record Buffer' when I attempt to modify a record containing a BLOB?

A: This error can occur for two different reasons.

Case 1 - BDE API Application
This error is caused by not setting up the record correctly. Make certain to call DbiInitRecord on the record buffer before reading the record from the table.

CASE 2 - Any BDE Application
This error occurs when scrolling through more than 64 blobs in the results of a dead query. The solution is to either make the results of the query a live result set, or somehow limitscrolling blobs. (Such as having your SELECT statement select fewer records.)

NOTE:A future release of the BDE is planned to be able to be configured to get past the present hard coded limit.

Q: Why does my application crash when using filters?

A: Beside the general suggestion at the top of this file, as well as errors in setting up offsets of the filter, this error can be caused by setting the iPriority parameter to DbiAddFiler to 0. Make certain to set this value to 1. (There is a misprint in some copies of the BDE Users Guide to set this to 0).

Q: What's new in the 32 bit version?

A: 1) Long filename support.
2) UNC filename support.
3) Transactions on Paradox and dBASE tables.
4) Cached Updates.
5) The method for preparing queries has changed, requiring the use of a new BDEfunction, DbiQAlloc, to obtain a statement handle.
6) BDE Configuration Information can now optionally be stored in the registry.
7) New SQL Engine.

Q: Why am I getting the error "Too many open files" in the 16

bit version?

A: You need to call SetHandleCount and make sure the value is larger than the MAXFILEHANDLES statement in BDECFG.

PARADOX TABLE SPECIFICS

Q: How do I access tables in a read-only directory (such as a table on a CD-ROM)?

A: A directory lock needs to be placed within the directory containing the table to prevent the BDE from attempting to create a lock file (.LCK) in that directory. An example of this is included in the SNIPIT example application, in the file RDOLOCK.C.

Q: I got "Table locked/busy" and it's neither?

A: This error happens when the application tries to lock a table when the corresponding prevent lock already exists, or when the application tries to set a prevent lock and a conflicting lock already exists. Check for old, unused lock files. Delete any .lck files that may exist after all BDE applications have terminated. Also run the TUtility program to test for table validity.

Q: Why do I get the error "Multiple Paradox Net Files Found?"

A: In addition to the case of two application referencing different physical PARADOX.NET files, this error message can be caused by old .LCK files or when two applications using the same NET file are mapped differently. Also make certain that the BDE DLL's are not left in memory. This error can also occur if differnt mapping are used to reference the same file. For example, if h:\one\two on one system points to the same physical location as h:\three on another system, the BDE will not recognize that it is only a single .NET file. Make certain that the path matches.

Q: I am working with a Peer-to-Peer network, like Windows for Workgroups, Lantastic, or Personal Netware, and I am getting "Multiple Paradox Net Files Found", what is wrong?

A: The BDE requires that everyone use the exact same directory and drive letter. Therefore, if the tables are on c:\tables on the shared drive and user maps the shared drive as f: then f:\tables points to the same place as the shared drive's c:\tables. However, the drive letter must also match. The way around this is to subst drive c: as drive f: This way both c: and f: point to the same drive. Now f:\tables on the shared drive's computer points to the c:\tables and everyone is using f:\tables. For more information on the subst command look in your DOS

manual. However, remember that if LASTDRIVE is set to f: in your CONFIG.SYS then the last drive you can subst is f:

Q. I am getting the error "Lock file has grown too large". What is the solution?

A. Download TI2993 for a detailed answer.

dBASE TABLE SPECIFICS

Q: How do I insert a Date into a dBASE table using Local SQL?

A: 'INSERT INTO myTable (myField) value ("01/01/94")'

Q: How can I create a case-insensitive index?

A: Case insensitive indexes are not directly supported, but can be simulated using an expression index with the upper() function.

Q: Why can I insert a duplicate record into a dBASE table when I have a unique index defined?

A: dBASE does not enforce the uniqueness within the table - only within the index. When the table is opened on that index, only one record with a given key value will be present in the index, but both values exist in the table.

Q: Why am I getting the error DBIERR_NOSUCHINDEX, "Index does not exist", when I attempt to open a FOX table?

A: The BDE currently does not transparently support FOX (CDX) indexes. When a FOX table has a CDX index, it is marked as the production index of that table. The BDE by design has to open the production index when the table is opened. As it cannot access the FOX production index (CDX), it returns as error. Currently, to open a FoxPro table with an index, you must use an ODBC driver. In a future release of the BDE native support for .CDX indexes is planned.

Delphi Quick Info Sheet

NUMBER : 2768
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : November 5, 1996

TITLE : Delphi Quick Info Sheet

Borland Delphi for Windows Quick Info Guide

RADical performance Windows development

Delphi is the only Windows development tool to combine the Rapid Application Development (RAD) benefits of visual component-based design, with the power of an optimizing native code compiler and scalable database access.

Delphi represents the next generation in:

- Performance--with the world's fastest compiler
- Rapid Application Development--via visual Two-Way-Tools
- Component Reuse--a true object-oriented environment Scalable Database Access--the fast track to client/server

The fastest way to the fastest applications

Delphi is the next-generation Windows development tool, combining the visual design environment with the unrivaled performance of a world-class optimizing native code compiler. Delphi includes the world's fastest compiler, capable of compiling Windows applications at more than 350,000 lines per minute, resulting in executable files (EXEs) that are robust and lightning-fast.

Delphi executables are also immediately deployable and completely royalty-free, and no runtime interpreter Dynamic Link Library (DLL) is required. Delphi applications typically execute up to 10 to 20 times faster than interpreted p-code.

Only Borland can offer this unique combination of integrated tools including its award-winning report writer, ReportSmith, the Borland Database Engine, and the Local InterBase Server.

RAD, from prototype to production

Delphi accelerates Windows development by providing a comprehensive suite of visual design and debugging tools that move you seamlessly from prototyping to deployment of finished applications. Delphi's Two-Way-Tools help you

optimize your programming time by switching effortlessly between visual design and the underlying source code.

Reuse components to maximize productivity

Included with Delphi is the comprehensive Visual Component Library (VCL), a collection of more than 75 standard Windows objects such as dialogs, buttons, and list boxes, along with a host of additional reusable objects including database controls, notebook tabs, grids, multimedia controls, and much more! Delphi's Integrated Development Environment (IDE) is fully customizable. Custom reusable components, external DLLs, and commercial VBXs can be rapidly added to Delphi's Component Palette.

Delphi is a true object-oriented development tool, including polymorphism, inheritance, and encapsulation. Delphi's component-based architecture allows you to seamlessly integrate DLLs, VBX controls, and OLE 2.0 servers into your applications.

Smooth scaling to client/server

Delphi has been designed to provide an easy path to the fast-growing market of client/server applications development. Transparent local SQL development is possible with Delphi, using the built-in Local InterBase Server, which enables you to rapidly develop high-performance ANSI SQL-92 compatible applications for standalone systems.

"Delphi is Visual Basic done right." --PC/Computing, February 1995

Delphi Product Information:

Product

- Delphi for Windows CD-ROM only. Estimated street price $350*
- Special Introductory Offer of $199.95 for first 90 days (3/1/95-5/31/95).
- $50.00 rebate coupon inside for Pascal owners. 3.5" disks available separately from Borland for $19.95 plus $5.00 shipping.

Contents

Delphi for Windows
Local InterBase Server
ReportSmith 2.5 PC version

Manuals:
Delphi User's Guide
Delphi Database Application Developer's Guide
Delphi Component Writer's Guide
Creating Reports
InterBase User's Guide

Target Audience

Programmers, VARs, Pascal developers, C++ developers, Visual Basic Pro developers, PowerBuilder developers, Paradox and dBASE developers.

Delphi Accessories:

Turbo Assembler

$99.95
3.5" HD disks

Description:
Whether you're writing time-critical systems or speeding up your Delphi applications, the Turbo Assembler ensures your applications are as robust and efficient as possible.

Buyers:
Windows developers who want to build fast applications or add fast code to existing applications. Programmers using Microsoft Assembler who want a way to do faster, more flexible assembly language programming.

Visual Solutions Pack

$99.95
3.5" HD disks

Description:
With the Visual Solutions Pack you can add spreadsheets, databases, word processing, graphics, and more to your Delphi applications with virtually no coding.

Buyers:
Windows developers using Delphi or any other development tool that supports VBX format custom controls.

Paradox 5.0 for Windows

$495
CD-ROM or 3.5" HD disks

Special limited-time upgrade price of $129.95 for suite owners of competing databases. $30 rebate for previous Paradox owners.

Description:
With Paradox you get relational database power made easy. DLLs and OLE 2.0 controls built with Delphi are completely compatible with your Paradox applications.

Buyers:
Windows database developers who want to build fast, efficient, database applications.

Borland Delphi support services

Fast Fax for Detailed Information: 1-800-408-0001
TechFax for Technical Information: 1-800-822-4269
Connections Developer Program: 1-800-353-2211
Free Install Support: (408) 461-9195
Credit Card Advisor Line: 1-800-330-3372

On-line Services
- On CompuServe, type GO BORLAND
- On BIX, type JOIN BORLAND
- On GEnie, type BORLAND
- Borland Download Bulletin Board Service: (408) 431-5096
- For other Assist Support Services: 1-800-523-7070

Attention Borland resellers
- For information on obtaining evaluation copy order forms: 1-800-408-0001
- To request collateral: (408) 431-1950
- For more information: (408) 461-9000

Minimum System Requirements
- Intel 386-based PC or higher
- Microsoft Windows 3.1 or later, 100%-compatible version
- 6Mb of extended memory or higher
- CD-ROM drive (3.5" floppy dissks available separately)
- 30Mb hard disk space

Satisfaction guaranteed!

You can buy Delphi for Windows with complete assurance. If for any reason you are not fully satisfied with your purchase, you can return it to Borland within 90 days. No questions asked!

*Scholar price available through authorized Borland Educational resellers. Call 1-800-847-7797.

Instructions for Running Delphi from CD-ROM

NUMBER : 2777
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Instructions for Running Delphi from CD-ROM

RUNNING DELPHI FROM THE CD-ROM

Delphi can be run from a CD-ROM directly. However, it is highly recommended that at least a minimum installatiion be seriously considered before opting for running Delphi from CD-ROM.

For a minimum installation, 33MB of available hard disk space is required.

Running from a CD-ROM, 9MB of available hard disk space is required.

To configure the system to run Delphi from the CD-ROM,

1. Copy all files found in the CD's \RUNIMAGE\WINDOWS directory to the Windows directory of the hard disk.

2. Copy all files found in the CD's \RUNIMAGE\WINDOWS\SYSTEM directory to the Windows SYSTEM directory of the hard disk.

3. Edit DELPHI.INI in a text editor like Windows Notepad. Look for the section titled [Globals]. Make entry for the PrivateDir setting that resides on a writeable disk, preferably a local hard disk.

For instance, if a directory on the C: drive where chosen called DELPRIV, the relevant section of DELPHI.INI would look as follows.

```
[Globals]
PrivateDir=C:\DELPRIV\
```

Now, Delphi can be run from the \RUNIMAGE\DELPHI\BIN directory of the CD-ROM.

MODIFYING COMPONENTS

To allow modification of the component library used by Delphi while running from a CD ROM an copy of the Component Library must exist on a writeable disk. Use the following steps as a guide.

1. Copy COMPLIB.DCL from the CD's \RUNIMAGE\DELPHI\BIN directory

to a desired directory on the hard disk.

2. Run Delphi.

3. Bring down the Options menu and select Open Library.

4. Specify the directory and file name for the COMPLIB.DCL copied to the hard disk.

Making your Delphi apps show minimized.

NUMBER : 2778
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Making your Delphi apps show minimized.

Q: When I select the "Run Minimized" option in Program Manager to attempt to make my Delphi application execute in a minimized state, the Delphi application seems to ignore the setting and run normally. Why is this, and how to I fix it?

A: Delphi's Application object creates a hidden "application window," and it is that window, rather than your main form, that is being sent the command to show minimized. To fix this, make your main form's OnCreate event handler look like this:

```
procedure TForm1.FormCreate(Sender: TObject);
{$IFDEF WIN32}              { Delphi 2.0 (32 bit) }
var
  MyInfo: TStartUpInfo;
{$ENDIF}
begin
{$IFDEF WIN32}              { Delphi 2.0 (32 bit) }
  GetStartUpInfo(MyInfo);
  ShowWindow(Handle, MyInfo.wShowWindow);
{$ENDIF}
{$IFDEF WINDOWS}            { Delphi 1.0 (16 bit) }
  ShowWindow(Handle, cmdShow);
{$ENDIF}
end;
```

In other words, for 16 bits, just pass cmdShow to ShowWindow. For 32 bits you need to obtain the start up info by calling the GetStartUpInfo procedure, which fills in a TStartUpInfo record, and then pass TStartUpInfo.wShowWindow to ShowWindow.

Delphi Client/Server and Power Builder Compared

NUMBER : 2779
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Delphi Client/Server and Power Builder Compared

An Evaluation of
Client/Server Development Tools:
A Comparison Between
Delphi Client/Server and PowerBuilder Enterprise

Prepared by Michael Lant, Sphere Data Systems

Table of Contents

Introduction: An Overview of Client/Server Tools

Client/server development tools remain one of the fastest-growing sectors of the software industry. Forrester Research predicts that the number of client/server developers will climb from 128,400 in 1994 to 698,000 in 1996, while META Group research

suggests that over 90% of all new applications will be based on the client/server model. This is an overwhelming endorsement of the popularity of client/server computing.

Client/server computing, however, is not the panacea that many have been predicting. Complex environments, with mixed data sources and immature development tools, have made client/server development a complex undertaking. Researchers such as the Gartner Group have found that the most pressing issues facing current developers of client/server applications include performance of deployed applications, rapid application development (RAD), component reuse and database scalability. Although these concerns are not unique to client/server application development, they are perhaps even more important in this sector, because of the well-documented limitations of first-generation client/server tools.

PCs and the Evolving Client/Server Model

PCs facilitated the downsizing era, by providing inexpensive platforms for the production of reasonably capable solutions that addressed genuine business needs. These applications brought with them the promise of freedom from host-based technologies and shortened development cycles. But as these PC- and network-based applications grew, user counts expanded and more functionality was necessary. PC database applications frequently "hit the wall."

In addition, all database applications have at their very heart business rules. These rules control processes such as creating new customers and data validation. Historically, PC application development has placed the enforcement of business rules into the user interface, so programmers would create a screen and write code to respond to user input. This code would perform a specific action based on the presence, or value of, certain information. Such actions might include allowing or preventing the posting of changes to customer data. But the user interface is certainly not the best place to store business rules.

The advent of client/server brought with it a new model that allows for business rules to be stored with the data itself. This is accomplished through the use of triggers and stored procedures created in the actual database. These procedures would enact routines to perform much the same processing that might be performed in a PC application, except that the host handles this task for all of its clients. This approach ensured that there was only one source for the business rules, since they were stored with the data. In many ways, this approach is enticing, but in reality it is plagued with problems.

When business rules are created to be enforced at the server level, they tend to be difficult to write and debug. According to David Sarna and George Febish in the September 1994 issue of Datamation: "These are not the tools for creating and maintaining good code. Doing it this way is SQL-abuse; SQL as a query language was never designed to do this kind of procedural

computation."

With this model, as an application grows, so do the number of rules that must be created and controlled by the database. Rather than focusing on the efficient storage and retrieval of data, server resources become tied down enforcing business rules - such as ensuring that a last name is supplied for each new customer. Again, Sarna and Febish point out that "many of your business and validation rules are needed so often that they must sit on the client, or performance will go straight to hell."

The Next Generation

Borland's Delphi is truly a next generation client/server development tool. Although it fully supports the placement of business rules in the user interface or on the server (or even a combination of both), it also presents the possibility of another model that separates the business rules from both the user interface and the server. This approach has the potential to provide greater performance, better scalability, better code reuse and an ideal platform for RAD. Delphi makes this possible because it employs an open architecture, whereby developers can use or create a wide variety of specialized components when creating applications.

Comparative reports of software tools typically focus on feature matrices or checklists to relate the features of one product versus those of its competitors - and in this context Delphi would certainly emerge as a winner. Checklists, however, do not necessarily help developers understand how one product could be far more effective than another. Thus, the content of this white paper is focused on the needs of client/server application developers, and how Delphi addresses those needs.

What is Delphi?

Delphi represents the next generation of client /server and general Windows application development tools. It is a product of Borland's long-standing leadership in programming languages, database development tools, and object-oriented programming (OOP) technology. Delphi is unique in providing a completely visual integrated development environment (IDE) that is built around a powerful object-oriented compiled language (Object Pascal). The IDE shields programmers from the complexity of Windows development, by encapsulating the Windows application programming interface (API) inside robust, reusable components.

The implementation of a component-based architecture enhances power and flexibility. Components range from simple items (such as buttons, text edit regions or sliders) through to powerful specialized tools (such as high-performance database components used for table access, queries and batch updates). While most client/server development tools include a limited selection of these basic components, Delphi ships with more than 75 reusable components.

Developers may modify Delphi components, create their own, or purchase third-party components (which include text processors, graphics, communications and statistical utilities, among others). Once installed on Delphi's component palette, these additional components become part of the IDE, so that Delphi can be tailored to whatever the developer (or team of developers) specifically requires.

Because of the unique way in which components are implemented in Delphi, it offers similar open architecture benefits to those of the ISA bus, for example. A component is a self-contained block of code that includes all functionality to work within a Delphi application. Just like inserting a new PC expansion card, any new component added to the library becomes part of Delphi and is indistinguishable from those that ship with the product. This open, component-based structure means that Delphi developers are not tied to any particular development methodology, and may use whatever tools or components are necessary to accomplish a task.
Redefining Client/Server Development

First generation visual client/server development tools (such as PowerBuilder) feature environments where all components are tightly integrated and dependent upon one another. Such an approach does not encourage modification or customization of existing components, because when a component is significantly altered, it has the potential of changing the way in which it interacts with others. This not only impedes the evolution of the base product, but it also has the potential to introduce unpredictable behavior.

Delphi, on the other hand, has as its foundation a true object-oriented language, with the result that its components are essentially autonomous objects. Component changes are therefore unlikely to affect overall product stability. Also, problems with any specific component behavior can often be corrected by the developer via subclassing, and thereby overriding the errant behavior. This also holds true for components created by a developer or purchased from third-party suppliers.

Delphi Component Types

Delphi's Visual Component Library (VCL) includes more than 75 pre-built components, which can be used in applications and deployed freely. These include the following:

Standard - Menus, Edit, Label, Buttons, Scroll Bar;

Additional - Bitmap Button, SpeedButton, Notebook, Outline, DrawGrid, Image, Multimedia; Data Access - Database, Table, Query, Batch Move, Report;

Data Controls - DataGrid, DBNavigator, DBLabel, DBEdit, DBImage, ComboBoxes;

Dialogs - Open, Save, Font, Color, Print, Print Setup,

Search/Replace, Tabbed Notebook;

System - Timer, FileListBox, Drive and Directory Combo OLE, DDE;

VBX - Chart, Picture, Gauge;

Samples - Gauge, ColorGrid, SpinButton, SpinEdit, Outline, Calendar.

Delphi developers can also create their own components, by modifying existing Delphi components or building them from scratch. The source code for Delphi's VCL is available to help developers in this task. Newly-developed components (or those purchased from third-party developers) can be added to Delphi's component palette.

Performance

In a client/server environment, overall performance depends upon a combination of factors, including efficiency of the database server, network bandwidth, "middleware" or driver performance, data access techniques, and speed of the user interface. A well-designed architecture should leverage all of these characteristics. In almost all respects, Delphi outperforms PowerBuilder by a significant margin, due to its optimizing compiler, efficient VCL components and high-performance native SQL drivers. Some sample benchmark results are listed below:

Operation (All times in seconds)	Delphi	PowerBuilder
String parse to measure non-database performance;reading file and splitting record into substrings	2.7	22.8
Use a query to load a form from a 20,000 record table. Form supports searching and filtering.	4.6	70.9
Post a record (single order)	1.4	1.3
Apply a filter on 20,000 record table	3.0	6.2
Update record	1.5	1.1
Search for a value, 20,000 records	1.1	1.5

User Interface Performance and Compiled vs Interpreted Code
The user interface (UI) is the most visible portion of any application. UI performance covers obvious functionality such as screen updates, along with the underlying processing of data (including numerical calculations and string manipulations). It is the portion that users interact with every day, and it is also the first place where performance differs noticeably between PowerBuilder and Delphi.

Delphi uses an optimizing native code compiler to generate standalone executable files (.EXEs) containing machine code instructions. These executable files are self-contained and, other than the database engine and the data itself, require no external files to run. Although PowerBuilder creates an executable file, it is not truly compiled. When a PowerBuilder application is run, source code instructions are converted to machine code before the instructions may be executed. This extra level of translation introduces significant run-time overhead and seriously degrades performance.

To support its runtime interpretation, PowerBuilder requires external support files in the form of runtime interpreter Dynamic Link Libraries (DLLs). In addition to the application code, these DLLs must be loaded into memory at runtime, creating a larger memory "footprint" and leaving less memory for processing and buffering data. As a result, Delphi applications execute between 10 and 20 times faster than applications created using p-code ("pseudo-code") interpreters such as PowerBuilder, even on target systems with minimal memory.

In moving from PowerBuilder 3.0a to 4.0, PowerSoft tuned its runtime interpreter to achieve slightly better performance on systems fitted with 16M of memory. This performance boost often comes at a cost, however, as final applications can increase from 20 to 70 percent in size. Despite minor enhancements, there is no way for an interpreted p-code language to even approach the execution speed of a compiled language like that of Delphi.

Data Access - A New Approach

The two primary methods of implementing client/server solutions with PowerBuilder involve placing business rules on either the client or the server. Unfortunately, neither one adequately addresses the issue of overall performance. If developers choose to place the burden of enforcing business rules on the server, they risk performance bottlenecks due to server and network overload. Yet shifting this task to the comparatively slow PowerBuilder UI is not an ideal solution. As it turns out, PowerBuilder's design makes it difficult not to attach at least some of the business rules directly to the UI.

Delphi, on the other hand, fully supports either of these techniques, as well as allowing for a more flexible model. Delphi developers can create a "virtual third-tier" layer containing business rules independent of the UI and the server. Instead, these rules can be encapsulated in non-visual components (NVCs) in a layer between the UI and database engine. Physically and conceptually, this separates the data and related rules from its representation to the user.

This approach brings the distributed computing vision of client/server another step closer to reality, and also provides a practical means of enhancing performance. All data moving to and from the application passes through this central conduit. The

"third-tier" NVCs may be single Delphi components such as a TTable or TQuery or sophisticated combinations of components. By having a single access point to lookup tables (such as State, Province or Product Codes that may be used by several forms), the application need only fetch this data once - when it is started. This can greatly reduce the number of server requests, alleviating network traffic and server load.

More importantly, developers can place connections to the main database tables here as well. As users move between forms, cursors to their data would remain in place. Thus, if basic customer information were displayed in one form, moving to another form displaying credit history would invoke it already pointing to the appropriate customer. PowerBuilder has limited functionality in this area through the dwShareData function, which allows sharing of data between DataWindows. This process results in duplication of large, complex components, and unless sufficient RAM is available, this can further degrade client performance. In the same situation, Delphi reduces the number of components used, since data access and display are accomplished with separate components.

Rapid Application Development (RAD)

Rapid Application Development (RAD) is an attempt to address the issues of building applications tailored to customer requirements and completing projects with minimal re-working. Historically, serious application development involved tremendous effort from a team of analysts who quizzed potential users to determine every possible requirement of the application. From this collection of information, volumes of carefully documented specifications were produced, then the users would sign off and programming commenced. Some time later, the development team would finally emerge with an application, and users were then expected to adopt the product.

This model had several shortcomings. Often, the people who built the application were not the same as those who did the analysis, resulting in reinterpretation of the specifications. Furthermore, users are often not well equipped to diagnose and describe their technology needs. During the development process, little communication existed between the developers and eventual users of the application. In days gone by, when terminals and batch operations were the only UI, this may have been an adequate model. However, with the advent of PCs, user expectations are significantly higher in regard to the UI characteristics of modern client/server applications. These applications must now provide access to data, and present it in ways that are most meaningful to users, with tools to assist in the analysis and interpretation of the data. As a result, most large organizations are confronted with a backlog of applications, which must be tackled with limited staff, budgets and time.

RAD is a development methodology based around an iterative process of specifying, creating and enhancing an application

until the final product is completed. RAD is based on the notion that it is difficult to fully predefine a problem domain as complex as today's typical business applications. As the developers step through each iteration from prototype to finished product, the requirements become more clearly defined. Although not ideal for batch-style applications, RAD is very effective for the development of applications where the UI design is a key consideration.

Minimum Coding, Maximum Flexibility

RAD development tools should allow developers to quickly build prototypes with as little code as possible, and Delphi is very powerful in this respect. Building a database application with a classic Customer -> Orders -> Details form (including database connections) can be done with no lines of code in Delphi. By comparison, PowerBuilder developers must write a dozen lines of code to simply link to a database, then they must also write code to open the form, query the tables and synchronize the tables appropriately.

Delphi's comprehensive array of data-aware components minimizes the level of customization that must be done to deliver a necessary feature or function. Where appropriate functionality does not exist in a VCL component, developers have the option of modifying the component or using a third-party component (such as a VBX control or Delphi native component). Any component may be subclassed (exploiting inheritance) and reused in other areas.

DataWindows vs Reusable Forms

RAD methodology assumes that application design evolves through several phases, so a true RAD tool should allow developers to easily adapt to changes (which can be significant). Because PowerBuilder combines data access and display in a single component, some design changes can prove extremely difficult. When a customer requirement suddenly dictates a new approach to either data access or display, this often requires a rewrite of the DataWindow and/or the entire form. Since Delphi separates these two concepts, the adaptation process is much simpler.

As with most first-generation visual development tools, PowerBuilder embeds application code inside objects - with the exception of the DataWindow, which represents both the conduit and the display format of the data. A DataWindow is created in a separate section (or "painter"), and cannot have event-handling code written until it is pasted into a form, whereupon the relevant application code is then permanently attached to that DataWindow. Since code is hidden inside objects scattered about the form, it becomes difficult for developers to locate, and it also may be affected by modifications to objects on the form. Furthermore, by "hard-wiring" the DataWindow to the code, it becomes difficult to reuse that DataWindow in other forms. Delphi developers address this issue by writing event handlers, which are pointers to specific procedures, or references within components that indicate what procedure will be called when a

particular event occurs (mouse click, timer interrupt, etc). Form procedures are stored in a "unit" file, with all the code visible to the developer, so there is no need to search through various objects track down specific code fragments. Once procedures are created, they may be called by any event of any component. Event handlers can also call procedures written for any other event handler, or even directly access components or procedures residing in other forms. Deleting an object (such as a button) does not delete procedures created for that object, so that code changes are easier, faster and safer.

Since Delphi uses the same property/event model for all components, whether they are visual or non-visual, it is more consistent and easier to learn than products such as PowerBuilder. The end result is that Delphi clearly provides an ideal environment for RAD.

Component Reuse

Inheritance is the ability to create new objects (descendants) from a base object (ancestor). A descendant inherits all characteristics of the ancestor, although these can be overridden. Inheritance is an object-oriented programming (OOP) fundamental that provides a means by which programmers can reuse code, and it encourages development of base objects with characteristics that are likely to be used again. Since descendants inherit any changes made to the parent object, changes that are made will propagate throughout any objects that descend from a base object. The resulting collection of objects is often called a class hierarchy or class library.

PowerBuilder allows developers to use inheritance to create a class library based on forms or groups of controls. The problem with the PowerBuilder approach is that it emphasizes inheritance of the user interface. Because DataWindows perform the task of both data access and data display, and because of the close binding between a DataWindow and various objects (and code) on a form, UI inheritance is the only type of inheritance that makes sense in PowerBuilder. However, what developers mostly need to preserve and reuse is not the visual representation of the data, but instead the complex logic that drives the business rules. In fact, the visual representation usually changes the most. Hence, reuse through inheritance in PowerBuilder often requires the developer to spend significant time overriding the UI in order to preserve the underlying code. As a result, PowerBuilder forms often carry around much unwanted baggage.

Delphi takes an entirely different approach. In its first release, Delphi, supports inheritance of forms in code only, compared to PowerBuilder's support of UI inheritance. With the ability to separate business rules from the UI, a Delphi developer need only write these once. Different forms can access the same data components and inherit their business rules, but are not bound to them. Hence the developer is free to create any

number of forms displaying data in whatever format is appropriate, knowing that the non-visual components supplying data to the user interface have fully encapsulated these business rules.

Such an approach would be difficult with a product such as PowerBuilder, because the underlying interpreter-based architecture is not truly object-oriented. According to Steve Benfield, quoted in the June/July 1994 issue of PowerBuilder Developers Journal: "PowerBuilder's DataWindows is not truly object-oriented. I could try to not use DataWindows and code PowerBuilder in a purely object-oriented way. However, the system would be so slow that it would be useless."

The Delphi Inheritance Model

Inheritance is fundamental to Delphi, and all VCL components ultimately descend from an object known as TObject (the "T" prefix is an accepted standard to denote an object type). TObject has a great deal of capability built into it, and knows how to handle certain types of errors, along with how to display itself in the form designer. Many other components descend from TObject, each acquiring more capability and becoming more specialized along the way.

The familiar case of copying to the Windows clipboard is an example of where Delphi's inheritance can be most powerful. This turns out to be a complex procedure requiring more than 100 lines of code in PowerBuilder, whereas the following code sample shows how the same CopyToClipBoard is handled in Delphi:

```
   if (( ActiveControl ) is TCustomEdit ) then
TCustomEdit(ActiveControl).CutToClipBoard
```

This examples surfaces three important issues. The first is that it highlights how some very basic functionality is missing from PowerBuilder. The second is that Delphi developers deal with functions such as this on a much higher level than with traditional programming languages. Thirdly, the inheritance benefits are obvious. TCustomEdit is an ancestor of most of the editable components in Delphi (such as TEdit, TDBEdit and so on) In this example, the is operator checks to see if the active component is a descendant of TCustomEdit, and if so, the next line casts the active control as its ancestor - TCustomEdit. Because CutToClipBoard is a method of TCustomEdit, all controls inherited from it will also inherit this method.

A DataWindows Alternative

DataWindows are an important feature of PowerBuilder. By attempting to incorporate all database capability into a single structure, the result is a component that is extremely powerful, but highly complex. The PowerBuilder 3.0a reference manual allocates almost 150 pages to explaining the properties and syntax of the dwDescribe and dwModify functions, which are just

two of the more than 130 functions used within DataWindows. According to Breck Carter, quoted in the June/July issue of PowerBuilder Developers Journal: "dwModify( ) and dwDescribe is a topic that I know has been a sticking point for many developers. These functions can be daunting with their screwy syntax and difficult-to-debug commands . . . If DataWindows' bazillion functions and events don't get you, you probably won't avoid being shocked and overwhelmed by dwModify."

PowerBuilder developers place all of their code and forms in a single .PBL application file, while Delphi applications reside in three main file types. The application itself comprises .DPR and .PAS files, while .DFM files contains the binary representation of forms (buttons, grids etc). Delphi's project manager provides a high-level view of all these files, and units from other directories or projects can be easily included. By storing units in separate files, team development is simplified and the allocation of tasks is made easier. Developers need not even be in the same physical location.

During compilation, Delphi keeps track of which files have changed since the last compile and only generates those changes. Delphi's compilation speed is the fastest in the industry, at more than 350,000 lines of code per minute on a 66MHz Pentium fitted with 8M of RAM. A reasonably complex client/server application containing over a dozen forms will compile all files in less than a minute.

PowerBuilder's interpreter-based design makes direct comparisons with Delphi's compiled performance unrealistic. PowerBuilder developers cannot exit the code painter until their code is syntactically correct, so that moving between different code areas is greatly restricted. PowerBuilder then generates p-code from its scripting language, along with a runtime executable file. This p-code generation stage takes about the same amount of time as the complete syntax checking and compilation to machine code of a Delphi application. The PowerBuilder application, however, still requires translation and decoding overhead each time it is executed.

As the complexity of applications and size of development teams grows, so does the need to maintain control of the process. Keeping track of the latest versions of source files is an important part of this process, and Delphi accomplishes this by providing a direct interface to PVCS, a leading version control utility. Delphi will directly link to the PVCS DLL and integrate it into the development environment.

PowerBuilder developers frustrated with limitations of its programming language are forced to leave the environment and must turn to languages such as C++ or Pascal. Once they leave PowerBuilder, they must abandon any of their custom class libraries, which are not supported by other products. Delphi does not present such obstacles, as it is built around a comprehensive

structured, object-oriented language featuring pass-through SQL commands and inline assembler. Since Delphi can encapsulate applications into DLL format, any Windows program (including PowerBuilder) can call a Delphi-generated DLL. In the same manner, Delphi applications can also access DLLs created using other development tools.

Exception Handling

When developing serious client/server applications, it becomes essential to write code that is able to deal with the possibility of any process failing. Again, according to Breck Carter in the 1994 Special Issue of PowerBuilder Developers Journal: "The PowerScript language is less than ideal when it comes to error handling, Most functions can diagnose errors, but with very few exceptions the default action is to ignore the error and proceed. There are no debugging switches or compiler directives to change this behavior to trap all errors and halt. With the exception of the global SystemError event, there is little language support for automatic error detection or handling."

This lack of support from the PowerScript language makes it difficult, if not impossible to create robust error-handling routines without writing enormous amounts of code. In contrast, Delphi introduces a new and more powerful error-handling methodology. To understand how important this new approach is, we must first look at how it is done.

Fundamental to Delphi is the concept of objects. Every control in Delphi descends ultimately from a base object control called TObject. As shown in the diagram above, TObject and hence, all of its descendants already knows about errors. This knowledge is passed to the programmer by means of objects. When an error occurs, Delphi instantiates an Exception object of that exception type. For example, if a divide by zero error occurs, an EDivideByZero object is created. All forms have built into them, the means of dealing with most errors. Rather than having to write code to test for every possible failure, the following construct is used:

```
try
      { statements }
  except
      on ESomething do { specific exception handling code } ;
            else {default exception-handling code } ;
  end;
```

Delphi also allows developers to create their own exception types as follows:

```
type
EpasswordInvalid = class(Exception) ;
```

This new user type exception is used simply as follows:

```
   if Password <> CorrectPassword then
raise EpasswordInvalid.Create(    Incorrect password entered' ) ;
```

These examples are trivial and meant only to display concepts. Delphi exception handling actually has capabilities far beyond this. For instance, Windows General Protection Failure (GPF) errors that are impossible to deal with in PowerBuilder applications may be handled via Delphi's EGPFault exception type. This simple construct provides enormous power and flexibility, and minimizes the amount of code that needs to be written, further shrinking the development cycle.

Scalability

As more attention is focused on client/server development, the scope of such applications is expanding. Where once it was sufficient to provide basic data entry and inquiry capabilities, users now demand that their applications have functionality far beyond what can be provided from mainframe solutions. Delphi is the ideal tool for this type of transition, being built around Object Pascal, an enhancement of a language that is already known by more than two million developers.

Delphi does not hide code within UI objects, and is thus more familiar to mainframe developers and easier to work with than PowerBuilder. It maximizes code reuse, minimizes development turnaround time and has the performance and stability necessary for large applications. It is also infinitely extensible, due to its OOP architecture, comprehensive array of components and full-featured programming language.

Many applications are also being scaled up from PC databases written in dBASE, Clipper, FoxPro and Paradox. As these applications make the transition from PC environments into the full client/server model, they must be able to address both PC and server data. PowerBuilder has extremely poor performance when accessing PC databases, and does not provide support for Paradox tables beyond the Paradox 3.5 format, nor does it fully support the Paradox file locking mechanism.

Some organizations decide that a portion of their data should reside on a server platform such as Oracle, while other portions should remain in a PC format such as dBASE. Because of Delphi's broad database support and extremely good performance against both PC and server data, it is the ideal tool for when mixed data access is required. Developers need only one development tool regardless of whether the data source is PC-based, server-based or both. Delphi is the only development tool that can adequately address all of these models. On the other hand, PowerBuilder has opted to largely ignore the PC database side.

Borland Database Engine

Although not component-based itself, the component-like

architecture of the Borland Database Engine (BDE) fits neatly into the architecture of Delphi. As the core technology for most of Borland's database tools including dBASE and Paradox, and used by millions of database users, the BDE has evolved into an extremely stable, high-performance technology that can deal with almost any database format It may be easily configured to access popular server formats (including Oracle, Sybase, Informix and InterBase) using high-performance SQL Links. PC database formats such as dBASE and Paradox are accessed directly via drivers built into BDE, which also includes support for the wealth of ODBC drivers.

The BDE is also sold as a separate product, as all of its functionality is accessible to developers via an API. However, Delphi developers are most likely to use the native data-aware components to interact with the BDE, although it is possible to write directly to the API or even bypass it altogether if required. This open architecture provides a level of power and flexibility not possible within PowerBuilder.

Local InterBase Server

The Local InterBase Server is a locally-installed Windows version of Borland's powerful InterBase workgroup server. Unlike the Watcom SQL engine that ships with PowerBuilder, this product is a full-featured SQL database, including:

Stored Procedures and triggers
Automatic two-phase commit
Explicit transaction management
Declarative referential integrity
Event alerters
Simultaneous access to multiple databases and transactions
Multi-generational architecture
Updatable views
Outer joins
User defined functions (UDFs)

Because the Local InterBase Server supports most of the functionality of the high-end InterBase server, developers can create SQL-based applications using locally-stored data. Thorough testing of these applications can be performed locally, without concern for either the server load or the possibility of damaging live data. Once an application has been tested and debugged locally, it may then be scaled to work with an external database server. If the developer has elected to use specific features of InterBase, these will perform identically when the application is deployed against an external InterBase server.

With PowerBuilder, because the Windows version of WATCH SQL lacks many of the features on the server versions of Watcom SQL, developers cannot experience the benefits of direct scaling that Delphi provides. When PowerBuilder accesses the Windows version of Watcom SQL, it must open a separate application window. Since this is a standard window, users can task-switch to the window or close it - either from the window itself or externally. This

makes the Windows version of Watcom SQL unsuitable in a production environment.

The Local InterBase Server, however, loads as a DLL in Delphi and cannot be accessed by users. This, combined with its comprehensive feature set, make the Local InterBase Server ideal for applications running on notebook PCs. When a user returns to the office, the application can be linked to the server version of the database. Applications may therefore scale from notebooks up to Windows NT or Novell NetWare NLM servers, and all the way up to the largest Unix-based systems, without developers needing to change a single line of code. By offering this capability, Delphi represents the ultimate in database scalability.

Conclusion

This paper has examined just a few of the many differences between Delphi and PowerBuilder. Both are clearly high end client/server development tools with excellent features. When examined from the perspectives of performance, scalability, reuse and RAD, Delphi clearly leads the way into the next generation of client/server development.

Moving to a tab by name on a TabSet

NUMBER : 2799
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Moving to a tab by name on a TabSet

Moving to a page by name on a TabSet.

Place a Tabset(TabSet1) and an Edit (Edit1) on
your form. Change the Tabset's Tabs Property in
the String List Editor to include 4 Tabs:

Hello,
World,
Of,
Delphi,

Change Edit1's onChange event to:

```
procedure TForm1.Edit1Change(Sender: TObject);
var
  I : Integer;
begin
  for   I:= 0 to tabset1.tabs.count-1 do
    if   edit1.text = tabset1.tabs[I] then
      tabset1.tabindex:=I;
end;
```

If You type any of the Tabs names in edit1 it
will focus on the appropriate tab.

Step by Step Configuration of an ODBC Driver

NUMBER : 2781
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Step by Step Configuration of an ODBC Driver

CONFIGURING AN ODBC DRIVER AND ALIAS FOR DELPHI

CONTROL PANEL'S ODBC OPTION

Delphi installs an ODBC option to the Windows Control Panel program. The ODBC option indicates the available data sources (drivers) installed for use by ODBC. As you will find by selecting the ODBC option, a number of formats are installed with Delphi and are seen in the main window titled Data Sources. Additional formats may be supported by the drivers installed and can be configured by selecting the Add... button.

If a new driver is to be added or removed,

1. Select the Drivers... button from the Data Sources Window. From the drivers dialog, select the Add... button and provide the path where the ODBC driver will be found.

2. Return to the Data Sources Windows and include the possible data sources available through the new driver by selecting its Add... button.

3. To configure options available for a particular data source use the Setup... button. The function of the Setup... button will vary with each data format. Very often options like the working directory for the driver are configured in this area.

Online help is available for each dialog involved with the ODBC option.

BDE CONFIGURATION UTILITY

After installing the ODBC driver, run the BDE Configuration utility to configure the database engine to use the new driver.

1. From the drivers page, select the New ODBC driver button.

2. A dialog titled Add ODBC driver will appear. The option for SQL link driver is what will distinguish the databases created using this ODBC driver.

3. Next select the default ODBC driver. Dropping down the list

from the combobox will reveal the file types supported by ODBC drivers installed on the system.

4. Select the default data source for the ODBC driver. Having set the ODBC driver in step 3 above, the list of this combobox will have the data source names appropriate for use with the selected driver.

5. Select Ok.

6. Returning to the drivers page, select File/Save from the main menu to save this configuration.

CREATING AN ALIAS IN THE DATABASE DESKTOP

While this can be done from the BDE Configuration utility, it is more convenient overall to create ODBC aliases from the Database Desktop.

1. From the File menu, select Aliases...

2. From the resulting Alias Manager dialog, select New.

3. Type the name for your new alias in the area labeled Database Alias.

4. Use the drop down list of the Driver Type combobox to select the driver appropriate for this alias. Paradox and dBase tables are considered STANDARD. If the ODBC driver was properly configured in the BDE Configuration utility it's name will appear in this list.

5. Additional options may appear depending upon the driver type you select.

6. When finished, select Keep New to store the new alias. Then select Ok. You will be prompted for whether or not to save the aliases to IDAPI.CFG. Select Okay.

The alias will now be usable from both the Database Desktop and Delphi.

Creating Database Aliases in Code

NUMBER : 2783
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Creating Database Aliases in Code

This Technical Information document will help step thru concepts regarding the creation and use of ALIASES within your Delphi Applications.

Typically, you use the BDE Configuration Utility BDECFG.EXE to create and configure aliases outside of Delphi. However, with the use of the TDatabase component, you have the ability to create and use this ALIAS within your application-- not pre-defined in the IDAPI.CFG.

The ability to create Aliases that are only available within your application is important. Aliases specify the location of database tables and connection parameters for database servers. Ultimately, you can gain the advantages of using ALIASES within your applications-- without having to worry about the existance of a configuration entry in the IDAPI.CFG when you deploy your application.

Summary of Examples:
------- -- ---------
Example #1:
Example #1 creates and configures an Alias to use STANDARD (.DB, .DBF) databases. The Alias is then used by a TTable component.
Example #2:
Example #2 creates and configures an Alias to use an INTERBASE database (.gdb). The Alias is then used by a TQuery component to join two tables of the database.
Example #3:
Example #3 creates and configures an Alias to use STANDARD (.DB, .DBF) databases. This example demonstrates how user input can be used to configure the Alias during run-time.

Example #1: Use of a .DB or .DBF database (STANDARD)

1. Create a New Project.
2. Place the following components on the form:
   - TDatabase, TTable, TDataSource, TDBGrid, and TButton
3. Double-click on the TDatabase component or choose Database Editor from the TDatabase SpeedMenu to launch the Database Property editor.
4. Set the Database Name to 'MyNewAlias'. This name will

serve as your ALIAS name used in the DatabaseName Property for dataset components such as TTable, TQuery, TStoredProc.

5. Select STANDARD as the Driver Name.
6. Click on the Defaults Button. This will automatically add a PATH= in the Parameter Overrides section.
7. Set the PATH= to C:\DELPHI\DEMOS\DATA (PATH=C:\DELPHI\DEMOS\DATA)
8. Click the OK button to close the Database Dialog.
9. Set the TTable DatabaseName Property to 'MyNewAlias'.
10. Set the TDataSource's DataSet Property to 'Table1'.
11. Set the DBGrid's DataSource Property to 'DataSource1'.

12. Place the following code inside of the TButton's OnClick event.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Tablename:= 'CUSTOMER';
  Table1.Active:= True;
end;
```

13. Run the application.

*** If you want an alternative way to steps 3 - 11, place the following code inside of the TButton's OnClick event.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Database1.DatabaseName:= 'MyNewAlias';
  Database1.DriverName:= 'STANDARD';
  Database1.Params.Clear;
  Database1.Params.Add('PATH=C:\DELPHI\DEMOS\DATA');
  Table1.DatabaseName:= 'MyNewAlias';
  Table1.TableName:= 'CUSTOMER';
  Table1.Active:= True;
  DataSource1.DataSet:= Table1;
  DBGrid1.DataSource:= DataSource1;
end;
```

*****

Example #2: Use of a INTERBASE database

1. Create a New Project.
2. Place the following components on the form:
   - TDatabase, TQuery, TDataSource, TDBGrid, and TButton
3. Double-click on the TDatabase component or choose Database Editor from the TDatabase SpeedMenu to launch the Database Property editor.
4. Set the Database Name to 'MyNewAlias'. This name will serve as your ALIAS name used in the DatabaseName Property for dataset components such as TTable, TQuery, TStoredProc.
5. Select INTRBASE as the Driver Name.
6. Click on the Defaults Button. This will automatically add the following entries in the Parameter Overrides section.

```
SERVER NAME=IB_SERVER:/PATH/DATABASE.GDB
USER NAME=MYNAME
OPEN MODE=READ/WRITE
SCHEMA CACHE SIZE=8
LANGDRIVER=
SQLQRYMODE=
SQLPASSTHRU MODE=NOT SHARED
SCHEMA CACHE TIME=-1
PASSWORD=
```

7. Set the following parameters

```
SERVER NAME=C:\IBLOCAL\EXAMPLES\EMPLOYEE.GDB
USER NAME=SYSDBA
OPEN MODE=READ/WRITE
SCHEMA CACHE SIZE=8
LANGDRIVER=
SQLQRYMODE=
SQLPASSTHRU MODE=NOT SHARED
SCHEMA CACHE TIME=-1
PASSWORD=masterkey
```

8. Set the TDatabase LoginPrompt Property to 'False'. If you supply the PASSWORD in the Parameter Overrides section and set the LoginPrompt to 'False', you will not be prompted for the password when connecting to the database. WARNING: If an incorrect password in entered in the Parameter Overrides section and LoginPrompt is set to 'False', you are not prompted by the Password dialog to re-enter a valid password.

9. Click the OK button to close the Database Dialog.
10. Set the TQuery DatabaseName Property to 'MyNewAlias'.
11. Set the TDataSource's DataSet Property to 'Query1'.
12. Set the DBGrid's DataSource Property to 'DataSource1'.

13. Place the following code inside of the TButton's OnClick event.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Query1.SQL.Clear;
  Query1.SQL.ADD(
        'SELECT DISTINCT * FROM CUSTOMER C, SALES S
        WHERE (S.CUST_NO = C.CUST_NO)
        ORDER BY C.CUST_NO, C.CUSTOMER');
  Query1.Active:= True;
end;
```

14. Run the application.

Example #3: User-defined Alias Configuration

This example brings up a input dialog and prompts the user to enter the directory to which the ALIAS is to be configured to.

The directory, servername, path, database name, and other neccessary Alias parameters can be read into the application from use of an input dialog or .INI file.

1. Follow the steps (1-11) in Example #1.
2. Place the following code inside of the TButton's OnClick event.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NewString: string;
  ClickedOK: Boolean;
begin
  NewString := 'C:\';
  ClickedOK := InputQuery('Database Path',
        'Path: --> C:\DELPHI\DEMOS\DATA', NewString);
  if ClickedOK then
  begin
    Database1.DatabaseName:= 'MyNewAlias';
    Database1.DriverName:= 'STANDARD';
    Database1.Params.Clear;
    Database1.Params.Add('Path=' + NewString);
     Table1.DatabaseName:= 'MyNewAlias';
    Table1.TableName:= 'CUSTOMER';
    Table1.Active:= True;
    DataSource1.DataSet:= Table1;
    DBGrid1.DataSource:= DataSource1;
  end;
end;
```

3. Run the Application.

See Also:

Delphi On-line help -->
Database Properties Editor
TDatabase

Creating & Deleting TFields at run-time

NUMBER : 2790
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 24, 1996

TITLE : Creating & Deleting TFields at run-time

TField components (or more appropriately, descendants of the TField component of types corresponding to field types) can be created at design-time using the Fields Editor. The Fields Editor is invoked by double-clicking on the design icon for a TTable or TQuery component. But TField descendants can also be created and deleted at run-time.

Descendants of the TField component (such as TStringField, TIntegerField, etc.) are created by invoking the Create method for the type of TField descendant appropriate to the field in the data set. That is, the Create method for the TStringField descendant of TField would be called to create a TField descendant for a string-type field in the current data set. The Create method requires one parameter, that of the owner of the TField descendant, which is the containing TForm. After creating the TField descendant component, a number of key properties need to be set in order to connect it with the field in the data set. These are:

FieldName: the name of the field in the table.
Name: a unique identifier for the TField descendant component.
Index: the TField descendant component's position in the array of TFields (the Fields property of the TTable or TQuery with which the TField will be associated).
DataSet: the TTable or TQuery with which the TField will be associated.

The code snippet below demonstrates creating a TStringField. The containing TForm is called Form1 (referred to here with the Self variable), the active data set is a TQuery named Query1, and the field for which the TStringField component is being created is a dBASE table field named CO_NAME. This new TField descendant will be the second TField in the Fields array property of Query1. Note that the data set with which the new TField descendant will be associated (in this case, Query1) must be closed prior to adding the TField and then reopened afterwards.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  T: TStringField;
begin
  Query1.Close;
  T := TStringField.Create(Self);
  T.FieldName := 'CO_NAME';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
```

The example above example creates a new TStringField named Query1CO_NAME.

Deleting an existing TField descendant is merely a matter of invoking the Free method for that component. In the example below, the TForm's FindComponent method is used to return a pointer to the TStringField component named Query1CO_NAME. The return value for the FindComponent will either be of type TComponent if successful or nil if unsuccessful. This return value can be used to determine whether the component actually exists prior to invoking its Free method.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TC: TComponent;
begin
  TC := FindComponent('Query1CO_NAME');
  if not (TC = nil) then begin
    Query1.Close;
    TC.Free;
    Query1.Open;
  end;
end;
```

As with creating a TField, if the data set associated with the TField descendant is currently active, it must be closed or deactivated prior to invoking this method.

How to iterate through the fields of a table

NUMBER : 2791
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to iterate through the fields of a table

Getting a list of the fields in a table at run-time can be as simple as a call to the GetFieldNames method of the TTable, TQuery, or TStoredProc component. The DetFieldNames method returns a list of the fields that comprise the structure of the data set in the form of a TStrings list, which may be inserted into such visual components as a TListBox through its Items property:

```
ListBox1.Clear;
Table1.GetFieldNames(ListBox1.Items);
```

Of course, the TStrings list returned by the GetFieldNames method need not be used with a visual component. It could just as well serve as an array of field names stored entirely in memory, that can be used as a list or array.

But it is also possible to retrieve much more information about the fields in a table than just the names. Other descriptive attributes include field types and sizes. Retrieving tyhese values is slightly more involved than the use of the GetFieldNames. Basically, this process involves iterating through the FieldDefs property of the TTable, TQuery, or TStoredProc component. The FieldDefs property is essentially an array of records, one record for each field in the structure. Each field record contains information about the field, including its name, type, and size. It is a relatively straightforward process to iterate through this array of field descriptions, extracting information about individual fields.

There are a number of reasons why a program might need to query the structure of a table used in the application. One reason is a prelude to creating TField components at run-time that represent the fields in the table. The information gleaned from the structure of the table form the basis of the TField components to be created.

The example below demonstrates how to iterate through the fields available in a TTable or TQuery. The example extracts information about the available fields and displays the information in a TListBox, but the same methodology can be used to provide information necessary for the dynamic building of TField descendants. The example uses a TTable as the data set, but a TQuery can be used in the same manner as both TTable and TQuery components incorporate the Field-Defs property the same way.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  F: TFieldDef;
```

```
  D: String;
begin
  Table1.Active := True;
  ListBox1.Items.Clear;
  with Table1 do begin
    for i := 0 to FieldDefs.Count - 1 do begin
      F := FieldDefs.Items[i];
      case F.DataType of
        ftUnknown: D := 'Unknown';
        ftString: D := 'String';
        ftSmallint: D := 'SmallInt';
        ftInteger: D := 'Integer';
        ftWord: D := 'Word';
        ftBoolean: D := 'Boolean';
        ftFloat: D := 'Float';
        ftCurrency: D := 'Currency';
        ftBCD: D := 'BCD';
        ftDate: D := 'Date';
        ftTime: D := 'Time';
        ftDateTime: D := 'DateTime';
        ftBytes: D := 'Bytes';
        ftVarBytes: D := '';
        ftBlob: D := 'BLOB';
        ftMemo: D := 'Memo';
        ftGraphic: D := 'Graphic';
      else
        D := '';
      end;
      ListBox1.Items.Add(F.Name + ', ' + D);
    end;
  end;
  Table1.Active := False;
end;
```

How to match file date/time stamps.

NUMBER : 2792
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to match file date/time stamps.

Q: "How can I write a function that sets the date of one file equal to the date of another file?"

A: No problem. Just use the following function, which takes two strings representing full DOS path/file names. The file who's date you wish to set is the second parameter, and the date you wish to set it to is given by the file in the first parameter.

```
procedure CopyFileDate(const Source, Dest: String);
var
  SourceHand, DestHand: word;
begin
  SourceHand := FileOpen(Source, fmOutput);          { open source file }
  DestHand := FileOpen(Dest, fmInput);               { open dest file }
  FileSetDate(DestHand, FileGetDate(SourceHand)); { get/set date }
  FileClose(SourceHand);                             { close source file }
  FileClose(DestHand);                               { close dest file }
end;
```

Adding Graphics in Your Listboxes and Comboboxes

NUMBER : 2793
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Adding Graphics in Your Listboxes and Comboboxes

TI: Inserting graphics in Owner Drawn ListBoxes and ComboBoxes

The ability to place graphics inside ListBoxes and ComboBoxes can improve the look of your application and set your user interface apart from the others.

Q: How do I stick graphics in a Listbox or ComboBox???

Here is an step-by-step example.....

1. Create a form.

2. Place a ComboBox and Listbox component on your form.

3. Change the Style property of the ComboBox component to csOwnerDrawVariable and the Style property of the ListBox to lbOwnerDrawVariable.

An Owner-Draw TListBox or TComboBox allows you to display both objects (ex. graphics) and strings as the items. For this example, we are adding both a graphic object and a string.

4. Create 5 variables of type TBitmap in the Form's VAR section.

5. Create a Procedure for the Form's OnCreate event.

6. Create a Procedure for the ComboBox's OnDraw Event.

7. Create a Procedure for the ComboBox's OnMeasureItem.

8. Free the resources in the Form's OnClose Event.

```
{START OWNERDRW.PAS}
unit Ownerdrw;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
```

```
  TForm1 = class(TForm)
    ComboBox1: TComboBox;
    ListBox1: TListBox;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure ComboBox1DrawItem(Control: TWinControl; Index: Integer;
      Rect: TRect; State: TOwnerDrawState);
    procedure ComboBox1MeasureItem(Control: TWinControl; Index: Integer;
      var Height: Integer);
    procedure ListBox1DrawItem(Control: TWinControl; Index: Integer;
      Rect: TRect; State: TOwnerDrawState);
    procedure ListBox1MeasureItem(Control: TWinControl; Index: Integer;
      var Height: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  TheBitmap1, TheBitmap2, TheBitmap3, TheBitmap4,
  TheBitmap5 : TBitmap;
implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheBitmap1 := TBitmap.Create;
  TheBitmap1.LoadFromFile('C:\delphi\images\buttons\globe.bmp');
  TheBitmap2 := TBitmap.Create;
  TheBitmap2.LoadFromFile('C:\delphi\images\buttons\video.bmp');
  TheBitmap3 := TBitmap.Create;
  TheBitmap3.LoadFromFile('C:\delphi\images\buttons\gears.bmp');
  TheBitmap4 := TBitmap.Create;
  TheBitmap4.LoadFromFile('C:\delphi\images\buttons\key.bmp');
  TheBitmap5 := TBitmap.Create;
  TheBitmap5.LoadFromFile('C:\delphi\images\buttons\tools.bmp');
  ComboBox1.Items.AddObject('Bitmap1: Globe', TheBitmap1);
  ComboBox1.Items.AddObject('Bitmap2: Video', TheBitmap2);
  ComboBox1.Items.AddObject('Bitmap3: Gears', TheBitmap3);
  ComboBox1.Items.AddObject('Bitmap4: Key', TheBitmap4);
  ComboBox1.Items.AddObject('Bitmap5: Tools', TheBitmap5);
  ListBox1.Items.AddObject('Bitmap1: Globe', TheBitmap1);
  ListBox1.Items.AddObject('Bitmap2: Video', TheBitmap2);
  ListBox1.Items.AddObject('Bitmap3: Gears', TheBitmap3);
  ListBox1.Items.AddObject('Bitmap4: Key', TheBitmap4);
  ListBox1.Items.AddObject('Bitmap5: Tools', TheBitmap5);

end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  TheBitmap1.Free;
  TheBitmap2.Free;
```

```
  TheBitmap3.Free;
  TheBitmap4.Free;
  TheBitmap5.Free;
end;

procedure TForm1.ComboBox1DrawItem(Control: TWinControl; Index: Integer;
  Rect: TRect; State: TOwnerDrawState);
var
  Bitmap: TBitmap;
  Offset: Integer;
begin
  with (Control as TComboBox).Canvas do
  begin
    FillRect(Rect);
    Bitmap := TBitmap(ComboBox1.Items.Objects[Index]);
    if Bitmap <> nil then
    begin
      BrushCopy(Bounds(Rect.Left + 2, Rect.Top + 2, Bitmap.Width,
                Bitmap.Height), Bitmap, Bounds(0, 0, Bitmap.Width,
                Bitmap.Height), clRed);
      Offset := Bitmap.width + 8;
    end;
    { display the text }
    TextOut(Rect.Left + Offset, Rect.Top, Combobox1.Items[Index])
  end;
end;

procedure TForm1.ComboBox1MeasureItem(Control: TWinControl; Index:
                                          Integer; var Height: Integer);
begin
  height:= 20;
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer;
  Rect: TRect; State: TOwnerDrawState);
var
  Bitmap: TBitmap;
  Offset: Integer;
begin
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);
    Bitmap := TBitmap(ListBox1.Items.Objects[Index]);
    if Bitmap <> nil then
    begin
      BrushCopy(Bounds(Rect.Left + 2, Rect.Top + 2, Bitmap.Width,
                Bitmap.Height), Bitmap, Bounds(0, 0, Bitmap.Width,
                Bitmap.Height), clRed);
      Offset := Bitmap.width + 8;
    end;
    { display the text }
    TextOut(Rect.Left + Offset, Rect.Top, Listbox1.Items[Index])
  end;
end;

procedure TForm1.ListBox1MeasureItem(Control: TWinControl; Index: Integer;
```

```
  var Height: Integer);
begin
  height:= 20;
end;

end.
{END OWNERDRW.PAS}

{START OWNERDRW.DFM}
object Form1: TForm1
  Left = 211
  Top = 155
  Width = 435
  Height = 300
  Caption = 'Form1'
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  PixelsPerInch = 96
  OnClose = FormClose
  OnCreate = FormCreate
  TextHeight = 16
  object ComboBox1: TComboBox
    Left = 26
    Top = 30
    Width = 165
    Height = 22
    Style = csOwnerDrawVariable
    ItemHeight = 16
    TabOrder = 0
    OnDrawItem = ComboBox1DrawItem
    OnMeasureItem = ComboBox1MeasureItem
  end
  object ListBox1: TListBox
    Left = 216
    Top = 28
    Width = 151
    Height = 167
    ItemHeight = 16
    Style = lbOwnerDrawVariable
    TabOrder = 1
    OnDrawItem = ListBox1DrawItem
    OnMeasureItem = ListBox1MeasureItem
  end
end
{END OWNERDRW.DFM}
```

Using OnHint Events Among Mulitiple Forms

NUMBER : 2796
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Using OnHint Events Among Mulitiple Forms

Using OnHint Among Multiple Forms

Delphi's Online Help and Visual Component Library Reference describe an example for processing TApplication's OnHint event. The example shows how a panel can be used to display hints associated with other components. As the example sets the Application's OnHint method in the Form's OnCreate event, a program involving more than one form will have difficulty using this technique.

Moving the assignment of OnHint from the Form's OnCreate event to its OnActivate method will allow different forms involved in the application to treat Hints in their own way.

Here is an altered form of the source code presented in the Online Help and the VCL Reference.

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Panel1: TPanel;
    Edit1: TEdit;
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure DisplayHint(Sender: TObject);
  end;

implementation

{$R *.DFM}

procedure TForm1.DisplayHint(Sender: TObject);
begin
  Panel1.Caption := Application.Hint;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  Application.OnHint := DisplayHint;
end;
```

Making the Enter key work like a Tab in a TDBGrid

NUMBER : 2798
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Making the Enter key work like a Tab in a TDBGrid

Code to make the <Enter>key act as the tab key while inside a grid.

This code also includes the processing of the <Enter> key for the entire application - including fields, etc. The grid part is handled in the ELSE portion of the code. The provided code does not mimic the behavior of the <Tab> key stepping down to the next record when it reaches the last column in the grid - it moves back to the first column - .

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
{ This is the event handler for the FORM's OnKeyPress event! }
{ You should also set the Form's KeyPreview property to True }
begin
  if Key = #13 then                                    { if it's an enter key }
    if not (ActiveControl is TDBGrid) then begin { if not on a TDBGrid }
      Key := #0;                                       { eat enter key }
      Perform(WM_NEXTDLGCTL, 0, 0);                    { move to next control }
    end
    else if (ActiveControl is TDBGrid) then         { if it is a TDBGrid }
      with TDBGrid(ActiveControl) do
        if selectedindex < (fieldcount -1) then   { increment the field }
          selectedindex := selectedindex +1
        else
          selectedindex := 0;
end;
```

Delphi vs Visual Basic

NUMBER : 2780
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Delphi vs Visual Basic

An Evaluation of
Rapid Application Development Tools for Windows:
A Comparison Between
Delphi and Visual Basic

Table Of Contents

Introduction

With the growing demands on Windows applications developers to create increasingly complex applications in less time, the evolution of Rapid Application Development (RAD) tools has become a crucial focus of the development community. First-generation RAD tools for Windows included application frameworks (such as OWL and MFC), DLL-based class libraries, and VBXs (Visual Basic custom controls).

The release of Borland's Delphi heralds a new generation of RAD tools that combines the power of traditional 3GL compilers with the ease of use and development speed of a 4GL environment. This white paper will contrast the approach of the new technology used by Delphi with that of Microsoft's Visual Basic.

Windows Visual Development

The earliest methodology for Windows applications programming was to code directly using calls to the Windows API. The API provided a crude mechanism for creating such items as menus and windows, leaving the developer with an enormous coding task even when creating a rudimentary application. These early programs were typically created with C compilers that were equally crude by today's standards. As a result, the expectations for what could reasonably be achieved by professional developers was severely limited by schedule and performance constraints. Early applications were also, for the most part, independent. In other words, they were generally not reliant on each other and did not share data or invoke other applications.

Appreciating the underlying potential of the Windows environment, the development community demanded better tools, to enhance their productivity and facilitate the creation of more sophisticated applications. As Windows evolved, inter-application communications using DDE and OLE was introduced, then more powerful application frameworks products such as Borland's OWL and Microsoft's MFC appeared, as well as third-party products such as zApp from Zinc and Island Systems' object-Menu.

These libraries were used to encapsulate the most common functions of Windows applications, as well as to leave room for expansion and customization by developers. Thus, a programmer could quickly create a window with a certain border style, make it modal and add a "Close" button, then invoke it with a single call. Further, the advent of C++ compilers for Windows allowed experienced developers to exploit the power of object-oriented technology. Developers now had the means to create complex applications within acceptable schedules.

Object-oriented languages allowed developers to create classes and override specific virtual functions, providing a direct path to building custom libraries. This generation of tools still had two major shortcomings:

1. Although productivity had been significantly increased, schedules for complex applications development were still quite lengthy. For example, the common development scenario would proceed as follows:
    a) some sample screens would be created in a prototype environment or a resource tool;
    b) marketing would critique the screen design and modifications would be made;
    c) the "final" screens would be integrated with code to complete the application;
    d) any changes that were requested often became complicated and painful, since the code and the screens were so closely linked.

2. The expertise required made Windows programming the sole domain of the experienced developer. In other words, the

extensive needs of the corporate community simply could not be met due to the requirement for significant programmer expertise.

Thus evolved the next stage of Windows development, characterized by 4GL visual design environments such as Microsoft Visual Basic and high-end products such as Powersoft's PowerBuilder. These environments provided a major step forward in user-friendly development, with high-level, reusable components called controls which introduced the concept of a "building block" approach to software development. Most of the application development effort could now occur within a visual design tool where the programmer would piece together an interface from a suite of available libraries. Customization of the components can be accomplished by modifying a corresponding collection of properties sheet. Any "work" to be done within the application is triggered via events that affect the interface components (mouse clicks, keyboard entries and so on). With these first-generation visual development tools, the specification of the actions to be taken on these events is defined using a Basic-like scripting language.

As proven by their broad popularity, these tools went a long way toward solving some of the problems of corporate developers. However, there remained a serious deficit in their capabilities, due to their reliance on the visual design process for creating the application and also their underlying interpreted languages.

As the demands of Windows applications buyers continued to grow, developers were stretching the limits of the existing technologies to create projects such as mission-critical client/server applications. Team development and software quality assurance issues were becoming prominent. Applications were being designed as a series of modules that would need to interact seamlessly, and capable of communicating with and invoking other applications. For example, a user may have a need to insert a graphic in the context of an application. The graphic would be found by accessing a database created by some other application potentially residing over a network on a remote system. All of this needed to be transparent to the application at hand, so that users need not be concerned with where the graphic came from or how it was created.

Specifically, the following issues thus became crucial to professional applications developers:

* Performance
* Rapid Application Development
* Component Reuse
* Database Scalability

The next logical step in this evolution is technology that combines the significantly enhanced productivity of modern RAD tools with the power and flexible architecture of proven 3GL compiled languages. The remainder of this paper will contrast how Delphi and Visual Basic address these four key criteria for a

robust Windows development system.

It is assumed that readers of this paper are familiar with RAD design concepts, but a detailed knowledge of Visual Basic or Delphi is not necessary in order to understand this paper.

Performance

Performance of deployed applications is a key issue in today's highly competitive software market. Particularly for large, distributed client/server applications, any shortfalls in execution speed become far more apparent, due to higher overall system demands.

Delphi is based upon Object Pascal (a significant extension of the popular Borland Pascal 7.0) whereas Visual Basic uses Microsoft Basic as its underlying language. Delphi's performance is significantly better, simply because it generates compiled executable files, while Visual Basic produces semi-interpreted code. That is, Delphi is built around an optimizing native code compiler instead of the slower interpreted p-code used by products such as Visual Basic. This results in Delphi applications executing 10 to 20 times faster than interpreted code. Delphi's intelligent linker also enables segment optimization, thereby reducing executable file size by as much as 30 percent, which enables faster loading and additional performance gains.

Delphi can compile standalone executable files (.EXEs) as well as reusable Dynamic Linked Libraries (DLLs). For the ultimate in execution speed, Delphi also allows professional programmers to go one step further by writing in-line assembler code, for direct control of the microprocessor.

Other areas in which Delphi displays considerable performance gains over Visual Basic is in database connectivity. The database layer of Visual Basic is implemented via ODBC, as opposed to the more efficient Borland Database Engine used in Delphi (and other core Borland development tools). However, Delphi also supports links to data via ODBC drivers. The high-performance native SQL Links supplied with Delphi Client/Server also outperform comparable Visual Basic SQL connectivity options.

Benchmarks

Delphi's superior performance over Visual Basic becomes immediately apparent when running a few simple benchmarks. Consider the following examples, where a database is filled with items of text representing lastname, firstname, phone and street information. The phone number field is filled with consecutive integers, then the database is re-read and filled with a global array of integers from the phone number field. Finally, the global array is sorted to become reverse-ordered using a comparatively slow bubble sort algorithm.

Similar code can be written in both Delphi and Visual Basic, with

the stages of the benchmarks summarized in the following code fragments:

VB - Fill

```
Sub btnFill_Click ()
Dim k As Integer
 MaxArray = EdArraySize.Text
 For k = 1 To MaxArray
   Data1.Recordset.AddNew
   Data1.Recordset("LastName") = "Smith " + Str(k)
   Data1.Recordset("FirstName") = "Joe " + Str(k)
   Data1.Recordset("Phone") = Str(k)
   Data1.Recordset.Update
 Next k
 Data1.Recordset.MoveLast
End Sub
```

VB - Read

```
Sub btnSearch_Click ()
Dim k As Integer
Dim n As Integer
Dim s As String
 Data1.Recordset.MoveFirst
 For k = 1 To MaxArray
    s = edPhone.Text
    n = Val(s)
    Call AppendArray(k, n)
    Data1.Recordset.MoveNext
 Next k
End Sub
```

VB - Sort

```
Sub btnSort_Click ()
 Dim j As Integer
 Dim k As Integer
 Dim tmp As Integer
 For j = 1 To MaxArray - 1
   For k = 1 To MaxArray - j
    If GlobArray(k) < GlobArray(k + 1) Then
       ' Swap GlobArray[k+] with GlobArray[k] ...
       tmp = GlobArray(k + 1)
       GlobArray(k + 1) = GlobArray(k)
       GlobArray(k) = tmp
     End If
    Next k
   Next j
End Sub
```

Delphi - Fill

```
procedure TForm1.Button4Click(Sender: TObject);
var
 k,err: integer;
 s: string;
begin
 val(edDBsize.Text,maxArray,err);
 for k:=1 to maxArray do
   with Table1 do
     begin
       str(k,s);
       Append;
       FieldByName('Lastname').AsString := 'NewGuy'+s;
       FieldByName('Firstname').AsString := 'Paul'+s;
       FieldByName('Phone').AsString := s;
       Post;
     end
end;
```

Delphi - Read

```
procedure TForm1.btnSearchTestClick(Sender: TObject);
var
 s: string;
 n,err,k: integer;
begin
 val(edDBsize.Text,MaxArray,err);
 Table1.First;
 for k:=1 to MaxArray do
   begin
     s := DBedPhone.EditText;
     val(s,n,err);
     AppendArray(k,n);
     Table1.Next;
   end;
end;
```

Delphi - Sort

```
procedure TForm1.btnSortArrayClick(Sender: TObject);
var
 j,k,tmp: integer;
begin
 for j:=1 to MaxArray-1 do
   for k:=1 to MaxArray-j do
     if GlobArray[k] < GlobArray[k+1] then
       begin
         { Swap GlobArray[k+] with GlobArray[k] ... }
         tmp := GlobArray[k+1];
         GlobArray[k+1] := GlobArray[k];
         GlobArray[k]     := tmp;
       end;
end;
```

Benchmark Results

The following table shows the results for database tables ranging in size from 100 to 4000 records. The test stages Fill, Read and Sort correspond the code sections described on the previous pages.
(All benchmark times are in seconds.)

D = Delphi
FL = Fill
RD = Read
X = x Sort
ST = Sort

# items	D FL	VB FL	D RD	VB RD	D ST	X	VB ST	D Total	X Total	VB Total
100	2	2	30	1	0	0	0	2	1.5	3
1000	16	70	6	23	1	22	22	23	5	115
2000	33	141	12	46	4	21	84	49	5.5	271
3000	50	227	17	69	8	23.6	189	76	6.4	485
4000	67	297	23	77	15	19.6	294	106	6.3	668

As can be seen from the results, the resulting Delphi-generated code outperformed the Visual Basic routines, especially in code-bound portions such as the Sort stage, by about 20 times faster. Delphi's database access functionality was also shown to be about five times more efficient than that of the Visual Basic code.

Rapid Application Development (RAD)

The other side of the performance issue relates to the speed of application development, which is crucial for programmers intent on ensuring the fastest time to market for their products. The RAD features of an environment are the key to establishing how easy it is for programmers to progress from initial design and

prototyping through to final implementation and deployment.

A modern RAD environment provides developers with several elements that significantly speed the development process over the traditional sequential coding approach. These include:

* A visual design environment;
* High-level building block components (often called "controls");
* Contextual access to code segments directly, via objects. In other words, homing in on the specific code relating to a particular object.

Under Windows, the structure of an application is frequently molded around its graphical user interface (GUI), with the behavior of the application triggered by various Windows messages or events. The methodology for RAD flows according to the following outline:

1) The developer creates an empty window or form to contain the application's interface components;

2) The developer selects a component from a pool of available components, which are generally displayed as an array of icons. Components are then placed and sized on the form;

3) Relevant properties are set or adjusted for each component, according to the application's requirements;

4) Code is written and "attached" to all relevant events for each component;

5) The application is run within the development environment;

6) The developer can then continue to modify the form design or underlying code until the final working application is completed.

Both Visual Basic and Delphi subscribe to this general methodology, making the products appear deceptively similar. However, there are several key enhancements that Delphi adds to this process, including:

* More built-in controls Enhanced icon layout, via a fully-customizable, multi-page (tabbed) component palette;

* Extensive gallery of extensible project templates and experts;

* Enhanced object placement capabilities;

* Enhanced modification of property lists;

* Two-way, synchronized code window;

* Shared event functions;

* Integrated graphical debugging and object inspection

Controls

Visual Basic custom controls are referred to as VBXs, and a limited selection is supplied with Visual Basic itself. Additional controls are sold by third-party manufacturers, although these not only cost additional money but also extend the overall learning curve, due to variations in product styles. To utilize controls for a tabbed folder, notebook, database grid or 3D list box, for example, Visual Basic owners must obtain third-party VBXs. Some of the controls supplied with Visual Basic suffer from memory and other limitations, making it necessary to purchase third-party alternatives.

Delphi's Visual Component Library (VCL) is a comprehensive suite of high- performance controls that support all standard Windows functionality, along with additional features such as tabbed folders, notebooks, database grids and 3D list boxes. Delphi also supports third-party VBXs, providing access to a wide range of third-party components.

Control Icon Array

The Visual Basic control display is an array of icons with pictorial representations that are not always intuitive. In other words, developers can be left searching for the Image Control, for example, amongst many other icons with a similar look. Developers must then place a control into the form to be certain as to its identity. The Visual Basic control icon array can quickly become unwieldy as additional third-party custom controls are acquired. Since these icons are organized as a configurable rectangle, developers working with a large set of VBX controls are forced to give up valuable screen space or sacrifice accessibility of some controls.

Delphi solves these component layout problems with several enhancements. Firstly, Delphi's component palette is organized with several tabbed notebook pages, displaying icons in a single-row, scrollable toolbar format. This keeps the display uncluttered, yet fully accessible. More importantly, Delphi's customization options allow full configuration of the grouping, placement and display of components, so that the environment can be fine-tuned to suit the working style of an individual or development team. To address the problem of obscure or similar-looking icons, Delphi offers "fly-by help" showing the purpose of the control associated with the icon as the cursor is dragged over it.

Templates and Experts

Delphi includes pre-built templates that make it easy to develop standard applications or complex components such as MDI windows, database forms, multi-pages dialog and dual list boxes. The architecture is fully extensible, allowing developers to easily

register their own custom templates and experts into the gallery.

Object Placement

Delphi facilitates visual design with features such as automatic object alignment, sizing and scaling, while Visual Basic supports placement only. Delphi's automatic alignment also speeds up the creation of aesthetic forms.

Property Lists

A subtle yet significant distinction between the two development tools can be seen in the means of accessing property lists. Visual Basic users access a pull-down selection of options for a particular property via an entry bar at the top of the list, so that changing several property items, requires selection of the item, clicking on the entry bar to make the change, then clicking on the next item, and so forth. Delphi provides pull-down lists that can be accessed directly alongside the property value, making for more efficient and intuitive modifications.

Code Window

Delphi's code editing window synchronizes all visual design representations with the underlying source code. In other words, as the application is constructed by dropping objects into a form, the corresponding bug-free code is generated simultaneously. There are no limitations, since the code is always accessible, and developers can instantly switch between the code editor and the visual design tools, allowing them to select the most efficient mode for each part of the project.

Debugging and Object Inspecting

Visual Basic provides program debugging capabilities such as variable watches and a call stack monitor. However, this functionality is limited in that it cannot break on a specific

condition, and the call stack is modal, so it cannot be viewed during the entire debug session.

Delphi provides a full-featured debugger with conditional breakpoints and a modeless call stack viewer. The debug window and viewers can be saved from session to session, allowing developers to create a comfortable custom environment. Delphi also includes a powerful object browser similar to that used within Borland C++, which provides a comprehensive display of code objects and classes - including the capability to trace object lineage (inheritance, children) and virtual procedures.

Component Reuse

One of the most significant advances in applications development methodology is the concept of creating an application from high-level components. By linking predefined building blocks, developers need only define the "glue" between objects that

specifies the unique qualities of an application, with the potential for substantial productivity gains. Although Delphi and Visual Basic both provide various ways to reuse and share components and code, Delphi again delivers a cleaner and simpler solution.

The issue of reuse can be viewed in three areas:

* shared event functions

* reusable functions

* reusable building blocks (components)

Shared Event Functions

A common problem encountered in Windows programs is how to share a function that is executed upon the occurrence of several Windows events. Although the implementation is similar in both Delphi and Visual Basic, the Delphi solution has some obvious advantages. In Visual Basic, shared functions must be placed in the local code file or in a global .BAS file if the function is to be shared. The problem with this is that the function is now global to the entire project. In contrast, Delphi allows the function to be placed in the local file or in a DCU (Pascal unit file) which must be explicitly referenced only by the files that use it.

Reusable Functions and Libraries

In Visual Basic, common functions or libraries are accomplished by use of a global .BAS file, which then makes the functions accessible to every file in the project. The disadvantage to this approach is that the shared functions must be global to every file. Alternatively, Visual Basic can take advantage of functions organized in a DLL, but DLLs (like VBXs) must be created by another development tool external to Visual Basic, which requires a different level of expertise and an additional learning curve. All libraries for Delphi can be created from within the Delphi environment. Pascal code is organized as units, and shared functions are accessible through a Pascal unit by simply referencing the "library" unit that contains the desired function. Delphi can also use and create high-performance Windows DLLs. Further, Delphi's underlying programming language allows developers to reuse and customize functionality within a class via subclassing (see further details of OOP methodology below).

Components

VBXs can be developed for Visual Basic with functionality that is usable across different projects, but a significant disadvantage of VBXs is the complexity involved in creating them. There is a detailed set of restrictions associated with creation of a VBX such that they cannot be created within Visual Basic itself. Instead, the most common method to create a VBX is to use a C/C++

compiler to create a DLL and then put a VBX "wrapper" around it. The advantage of this is the speed of computation gained by using optimized C/C++ compiled code over the Visual Basic's interpreted technology. The disadvantage is that developers are forced to "switch gears" in order to work with the compiler, and the added complexity can lead to additional troubleshooting and debugging time.

Delphi components are more easily created. Unlike Visual Basic, where VBXs must be built using an external compiler, Delphi components are built within the Delphi development environment itself. This is an important distinction, because

professional developers prefer to work with a consistent set of tools. Being able to use Delphi to create reusable components becomes a major productivity enhancement, enabling more rapid development with the added benefits of reusability.

Additionally, since Delphi components are created with Delphi's optimizing native code compiler and linker, there is improved performance over traditional VBXs. One other considerable advantage of Delphi components is that developers can subclass the functionality of a component to create their own custom versions. If a specific VBX is insufficient for a Visual Basic user, the only alternative is to build (or purchase) an alternative VBX.

Programming Language

An obvious difference between Visual Basic and Delphi is the underlying programming language. The use of Object Pascal within Delphi has several important repercussions:

* Pascal is a more powerful and structured language than Basic.

* Object Pascal is a true object-oriented programming language, providing the benefits of inheritance, encapsulation and polymorphism;

* Pascal is a compiled language, ensuring high-performance executables;

* The organization of files as DCUs provides a cleaner mechanism for creating libraries of reusable code (see Shared Event Code);

* Object Pascal utilizes the world's fastest commercial compiler technology;

* Object Pascal support in-line assembler code for maximum performance;

One final point of differentiation is that in Visual Basic, all code files must be specifically associated with a form, except for a global .BAS file. In other words, a function must be

global to the entire project unless associated with a form. In Delphi, however, code files (and therefore classes and functions) can be disassociated with any form, allowing proper scoping of functions without any loss of functionality.

OOP Design Methodology

The power and flexibility of an object-oriented design methodology is widely accepted as the best way to solve complex, real-world programming problems. Object-oriented design provides both a solid foundation and elegant architecture for an application. Some of the benefits of OOP are:

* Shorter development cycles;

* Code that is highly maintainable;

* Code that is easily shared with other modules or other projects;

* Facilitation of team programming and version control;

* By exploiting object inheritance and polymorphism, the coding process can become much simpler and the code itself significantly more coherent;

* Applications can incorporate several functions that are mostly similar but have certain distinct "personality traits".

Object Pascal is a structured, object-oriented programming language, providing full support for class architectures, inheritance, virtual functions and polymorphism. Visual Basic is not an object-oriented language. Note that although developers need not be familiar with object-oriented concepts to create programs using Delphi, professional programmers will appreciate the benefit of these capabilities.

Database Scalability

A good RAD environment must address the pervasive issue of creating a database application, and Visual Basic and Delphi are no exceptions. In Visual Basic, developers can place a database component onto a form which can then have a property set that allows it to bidirectionally communicate with an ODBC-compatible database. The database component can be used as a crude mechanism to navigate through the database using arrows representing first, next, previous and last records. SQL queries can also be defined in code, to form a query snapshot into the database for viewing or computation. Crystal Reports is shipped with Visual Basic, providing a report generating capability. Setup of the database structures, the associated forms, interaction between them and most of the navigation through the database must all be done explicitly via the visual design tools or within code.

Delphi includes extensive database support including the Borland

Database Engine (BDE) for Paradox and dBASE access, and middleware layers that support local and remote SQL data access. The Borland database architecture provide developers with high-performance access to a variety of data sources including ODBC drivers. Delphi includes data access components and data-aware user interface components to provide a comprehensive database solution.

Delphi ships with several controls for data entry and display, including tables and grids. The grid control (TDBGrid Component) can be used to build a spreadsheet-style of application. A unique characteristic of the database grid control is that it can be linked to multiple database sources.

Delphi also includes wizards and experts that facilitate rapid design and implementation of databases and the corresponding user interface. The DataSet designer facility included with Delphi allows developers to rapidly create table or query data for database components. It is a simple matter to specify which set of fields from the database must be incorporated into the table or query.

When designing a database grid, an application often needs an editor to allow in-place modification of field data. Delphi's DBEdit provides a consolidated component to handle this task. Grid-aware, specialized versions of the control are available for labels, lists, combo boxes, images, memo (multi-line editors), check and radio buttons, lookup lists and lookup combo lists.

Delphi also features built-in support for queries and reports. A query control (TQuery Component) provides the ability to perform SQL queries in order to form the data set corresponding to the filtered elements of a database. If this data was extracted from dBASE or Paradox, you would also have the ability to modify, insert or delete records. By placing this component into a form that also contains the database component, developers can create a filtered, printable report based on some SQL or query into the data set. Delphi includes the award-winning ReportSmith report writer for PC and SQL databases. ReportSmith provides an intuitive interface for report creation using live data at design time, and it supports queries, crosstabs, templates, calculations and unlimited report sizes.

Moving Up to Delphi

Visual Basic developers who may be considering migrating their applications to Delphi, you may be concerned about the effort required to migrate existing Visual Basic applications in order to continue project development and maintenance within Delphi. Project migration is actually a fairly straightforward process. A conversion utility is available from EarthTrek, Inc (617) 273-0308 that performs most of the translation including project files, form files and code translation. The utility completes all of the possible automatic translation leaving some ambiguous language elements to be identified by a simple syntax check using the Delphi compiler. Many projects can be translated with

virtually no effort. Others may require a few hours of post-work to complete.

Conclusion

When examining the various RAD products in the marketplace, both Visual Basic and Delphi stand out as leading edge products. However, Delphi has clearly emerged as a next generation tool with its higher performance, highly facilitated visual design capability, extensive support for reusable components and database readiness. Delphi achieves its goals as a powerful application development system by combining a state-of-the-art visual design environment with the power, flexibility and reusability of a fully object-oriented language, the world's fastest compiler, and leading-edge database technology. Further, through the integration of Object Pascal, Delphi empowers developers with a full-featured programming environment without sacrificing rapid visual development thus allowing construction of sophisticated client-server applications in record time.

Calling a Delphi DLL from C

NUMBER : 2800
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Calling a Delphi DLL from C

Calling a DLL from your C Code.

First create a simple DLL in Delphi:

```
{ Begin DLL code }

Library MinMax;

Function Min(X, Y: Integer): Integer; export;
begin
  if X < Y then Min := X else Min := Y;
end;

Function Max(X, Y: Integer): Integer; export;
begin
  if X > Y then Max := X else Max := Y;
end;

Exports
  Min index 1,
  Max index 2;

begin
end.

{ End DLL code }
```

Then to Call it from your C Code:

1. In your .DEF File add:

```
IMPORTS
      Min   =MINMAX.Min
      Max   =MINMAX.Max
```

2. In your C application, you must prototype the functions as:

```
int FAR PASCAL Min(int x, y);
int FAR PASCAL Min(int x, y);
```

3. Now you can call Min or Max anywhere in your application.

pertains.

Getting a query's Memo field as a string

NUMBER : 2801
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Getting a query's Memo field as a string

Returning a Memo value to an Edit field without
using a memo field.

Place a query object on your form (Query1)
Place a Edit object on your form (Edit1)
Place a Button object on your form (Button1)
Double-Click on the query and add the memo field.
(Biolife.db using notes field)
Set Query1's SQL property to: Select * from Biolife
Set Query1's Active property to: True
Add the following code to Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bs : TBlobStream;
  p  : array [0..50] of char;
begin
  FillChar(p, SizeOf(p), #0);
  bs:= TBlobStream.Create(Query1Notes, bmRead);
  try
    bs.Read(p,50);
  finally
    bs.Free;
  end;
  Edit1.Text:=StrPas(p);
end;
```

How to encrypt a String

NUMBER : 2803
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to encrypt a String

Here is a program that demonstrates routines for encryption and decryption of strings. Note: We claim no responsibily for the security of these functions.

{ Begin code }

```
program Crypt;


uses WinCRT;

const
  C1 = 52845;
  C2 = 22719;

function Encrypt(const S: String; Key: Word): String;
var
  I: byte;
begin
  Result[0] := S[0];
  for I := 1 to Length(S) do begin
    Result[I] := char(byte(S[I]) xor (Key shr 8));
    Key := (byte(Result[I]) + Key) * C1 + C2;
  end;
end;

function Decrypt(const S: String; Key: Word): String;
var
  I: byte;
begin
  Result[0] := S[0];
  for I := 1 to Length(S) do begin
    Result[I] := char(byte(S[I]) xor (Key shr 8));
    Key := (byte(S[I]) + Key) * C1 + C2;
  end;
end;

var
  S: string;
begin
  Write('>');
  ReadLn(S);
  S := Encrypt(S,12345);
  WriteLn(S);
  S := Decrypt(S,12345);
```

```
  WriteLn(S);
end.
```

Undo in a Memo field

NUMBER : 2804
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Undo in a Memo field

Doing an UnDo in a Memo Field:

If you have a pop-up menu in a TMemo, and put shortcuts on it for the Cut,Copy, Paste, then you can handle those events, and call CuttoClipBoard, CopytoClipBoard, etc.

However, if you put an Undo option onto your pop-up menu (normally Ctrl-Z) how do you instruct the TMemo to do the Undo? If the built-in undo is sufficient, you can get it easier than a Ctrl+Z:

```
Memo1.Perform(EM_UNDO, 0, 0);
```

To check whether undo is available so as to enable/disable an undo menu item:

```
Undo1.Enabled := Memo1.Perform(EM_CANUNDO, 0, 0) <> 0;
```

Getting the Line number in a memo Field

NUMBER : 2805
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Getting the Line number in a memo Field

How do you figure out what line number you are currently on with a TMemo control?

The trick is to use the em_LineFromChar message. Try this:

```
procedure TMyForm.BitBtn1Click(Sender: TObject);
var
  iLine : Integer ;
begin
    iLine := Memo1.Perform(em_LineFromChar, $FFFF, 0);
    { Note: First line is zero }
    messageDlg('Line Number: ' + IntToStr(iLine), mtInformation,
                [mbOK], 0 ) ;
end;
```

Loading Bitmaps Into dBASE And Paradox BLOB Fields

NUMBER : 2807
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Loading Bitmaps Into dBASE And Paradox BLOB Fields

There are a number of ways to load a bitmap image into the BLOB field of a dBASE or Paradox table. Three of the easier methods involve 1) copying the data from the Windows clipboard into a TDBImage component connected to the BLOB field, 2) using the LoadFromFile method of the TBLOBField component, and 3) using the Assign method to copy an object of type TBitmap into the Picture property of a TBDBImage.

The first method, copying the bitmap from the clipboard, is probably most handy when an application needs to add bitmaps to a table when the end-user is running the application. A TDBImage component is used to act as an interface between the BLOB field in the table and the image stored in the clipboard. The PasteFromClipboard method of the TDBImage component is invoked to copy the bitmap data from the clipboard into the TDBImage. When the record is posted, the image is stored into the BLOB field in the table.

Because the Windows clipboard can contain data in formats other than just bitmap, it is advisable to check the format prior to calling the CopyFrom-Clipboard method. To do this, a TClipboard object is created and its Has-Format method is used to determine if the data in the clipboard is indeed of bitmap format. Note that to use a TClipboard object, the Clipbrd unit must be included in the Uses section of the unit that will be creating the object.

Here is an example showing the contents of the clipboard being copied into a TDBImage component, if the contents of the clipboard are of bitmap format:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  C: TClipboard;
begin
  C := TClipboard.Create;
  try
    if Clipboard.HasFormat(CF_BITMAP) then
      DBImage1.PasteFromClipboard
    else
      ShowMessage('Clipboard does not contain a bitmap!');
  finally
    C.Free;
  end;
end;
```

The second method of filling a BLOB field with a bitmap involves loading the bitmap directly from a file on disk into the BLOB field. This method

lends itself equally well to uses at run-time for the end-user as for the developer building an application's data.

This method uses the LoadFromFile method of the TBLOBField component, the Delphi representation of a dBASE for Windows Binary field or a Paradox for Windows Graphic field, either of which may be used to store bitmap data in a table.

The LoadFromFile method of the TBLOBField component requires a single parameter: the name of the bitmap file to load, which is of type String. The value for this parameter may come from a number of sources from the end-user manually keying in a valid file name to the program providing a string to the contents of the FileName property of the TOpenDialog component.

Here is an example showing the use of the LoadFromFile method for a TBLOBField component named Table1Bitmap (a field called Bitmap in the table associated with a TTable component named Table1):

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1Bitmap.LoadFromFile(
    'c:\delphi\images\splash\16color\construc.bmp');
end;
```

The third method uses the Assign method to copy the contents of an object of type TBitmap into the Picture property of a TDBImage component. An object of type TBitmap might be the Bitmap property of the Picture object property of a TImage component or it may be a stand-alone TBitmap object. As with the method copying the data from the clipboard into a TDBImage component, the bitmap data in the TDBImage component is saved into the BLOB field in the table when the record is successfully posted.

Here is an example using the Assign method. In this case, a stand-alone TBitmap object is used. To put a bitmap image into the TBitmap, the LoadFromFile method of the TBitmap component is called.

```
procedure TForm1.Button3Click(Sender: TObject);
var
  B: TBitmap;
begin
  B := TBitmap.Create;
  try
    B.LoadFromFile('c:\delphi\images\splash\16color\athena.bmp');
    DBImage1.Picture.Assign(B);
  finally
    B.Free;
  end;
end;
```

Searching your application's help file

NUMBER : 2818
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Searching your application's help file

The following code demonstrates how to bring up the WinHelp "Search" dialog for your application's help file. You can use TApplication's HelpCommand method to send the Help_PartialKey command to the WinHelp system. The parameter for this command should be a PChar (cast to a longint to circumvent typechecking) that contains the string on which you'd like to search. The example below uses an empty string, which invokes "Search" dialog and leaves the edit control in the dialog empty.

```
procedure TForm1.SearchHelp;
var
  P: PChar;
begin
  Application.HelpFile := 'c:\delphi\bin\delphi.hlp';
  P := StrNew('');
  Application.HelpCommand(Help_PartialKey, longint(P));
  StrDispose(P);
end;
```

Extracting A Bitmap From A BLOB Field

NUMBER : 2810
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Extracting A Bitmap From A BLOB Field

Extracting a bitmap from a dBASE or Paradox blob field -- without first saving the bitmap out to a file -- is a simple process of using the Assign method to store the contents of the BLOB field to an object of type TBitmap. A stand-alone TBitmap object or the Bitmap property of the Picture object property of a TIMage component are examples of compatible destinations for this operation.

Here is an example demonstrating using the Assign method to copy a bitmap from a BLOB field into a TImage component.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Picture.Bitmap.Assign(Table1Bitmap);
end;
```

In this example, the TBLOBField object Table1Bitmap is a BLOB field in a dBASE table. This TBLOBField object was created using the Fields Editor. If the Fields Editor is not used to create TFields for the fields in the table, the fields must be referenced using either the FieldByName method or the Fields property, both part of the TTable and TQuery components. In cases where one of those means is used to reference the BLOB field in a table, the field reference must be type-cast as a TBLOBField object prior to using the Assign method. For example:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Picture.Bitmap.Assign(TBLOBField(Table1.Fields[1]));
end;
```

A bitmap stored in a BLOB field may also be copied directly to a stand-alone TBitmap object. Here is an example showing the creation of a TBitmap object and storing into it a bitmap from a BLOB field.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  B: TBitmap;
begin
  B := TBitmap.Create;
  try
    B.Assign(Table1Bitmap);
    Image1.Picture.Bitmap.Assign(B);
  finally
    B.Free;
  end;
end;
```

Bitmaps And InterBase BLOB Fields

NUMBER : 2811
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Bitmaps And InterBase BLOB Fields

dBASE and Paradox tables provide a BLOB field to store binary data that, when the stored data is of bitmap-format, work as-is with the TDBImage component to display images. In Database Desktop, these field types are listed as Binary and Graphic (for dBASE and Paradox tables, respectively). However, the process of storing bitmap images in InterBase BLOB fields and using the stored data with TDBImage components is not as straight-forward.

InterBase tables do not have just one type of BLOB field. There are three variants, or sub-types: type 0, type 1, and user-defined sub-types. Types 0 and 1 are pre-defined types. Type 0 BLOB fields (the default type) are for storing general binary data. Type 1 BLOB fields are for storing text BLOB data. Either the pre-defined type 0 or a user-defined BLOB sub-type will allow the automated retrieval of bitmap data from the BLOB field that is to be displayed in a TDBImage component. Type 0 BLOB fields may be used for storing bitmap-format data or raw binary data. Here is an example of manually extracting bitmap data stored in a type 0 BLOB field (Table1-BLOBField) and displaying the data in a (non-data-aware) TImage component:

```
procedure TForm1.ExtractBtnClick(Sender: TObject);
begin
  Image1.Picture.Bitmap.Assign(Table1BLOBField);
end;
```

This manual method may be used or, more commonly, a data-aware control would be used so that the display of a given record's bitmap (in a BLOB field) will be automatically displayed. The TDBImage serves this purpose, and by setting the DataSource property to the TDataSource component associated with the underlying table and setting the DataField to the BLOB field containing the bitmap, the TDBImage component will display the image stored in the BLOB field and automatically load each record's BLOB field contents as the record pointer is changed.

The Database Desktop utility will allow the creation only of type 0 binary BLOB fields, no provision was made for user-defined BLOB field sub-types. If it is desired that a user-defined BLOB sub-type be used to store the bitmap data, it would need to be created with an SQL statement. Typically this would be through the WISQL utility, but an appropriate SQL statement in a TQuery would suffice. Here is an SQL statement that creates a table with a user-defined BLOB field sub-type:

```
CREATE TABLE WITHBMP
(
FILENAME CHAR(12),
BITMAP    BLOB SUB_TYPE -1
```

)

Once a table with a compatible BLOB field is created, storing bitmap data to the BLOB field and displaying the bitmap images in a TDBImage component uses the same methods as would be used with dBASE or Paradox tables.

There are a number of ways to load a bitmap image into a BLOB field. Three of the easier methods involve 1) copying the data from the Windows clipboard into a TDBImage component connected to the BLOB field, 2) using the LoadFromFile method of the TBLOBField component, and 3) using the Assign method to copy an object of type TBitmap into the Picture property of a TBDBImage.

The first method, copying the bitmap from the clipboard, is probably most handy when an application needs to add bitmaps to a table when the end-user is running the application. A TDBImage component is used to act as an interface between the BLOB field in the table and the image stored in the clipboard. The PasteFromClipboard method of the TDBImage component is invoked to copy the bitmap data from the clipboard into the TDBImage. When the record is posted, the image is stored into the BLOB field in the table.

Because the Windows clipboard can contain data in formats other than just bitmap, it is advisable to check the format prior to calling the CopyFrom-Clipboard method. To do this, a TClipboard object is created and its Has-Format method is used to determine if the data in the clipboard is indeed of bitmap format. Note that to use a TClipboard object, the Clipbrd unit must be included in the Uses section of the unit that will be creating the object.

Here is an example showing the contents of the clipboard being copied into a TDBImage component, if the contents of the clipboard are of bitmap format:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  C: TClipboard;
begin
  C := TClipboard.Create;
  try
    if Clipboard.HasFormat(CF_BITMAP) then
      DBImage1.PasteFromClipboard
    else
      ShowMessage('Clipboard does not contain a bitmap!');
  finally
    C.Free;
  end;
end;
```

The second method of filling a BLOB field with a bitmap involves loading the bitmap directly from a file on disk into the BLOB field. This method lends itself equally well to uses at run-time for the end-user as for the developer building an application's data. This method uses the Load-FromFile method of the TBLOBField component, the Delphi representation of an InterBase BLOB field.

The LoadFromFile method of the TBLOBField component requires a single parameter: the name of the bitmap file to load, which is of type String. The value for this parameter may come from a number of sources from the end-user manually keying in a valid file name to the program providing a string to the contents of the FileName property of the TOpenDialog component.

Here is an example showing the use of the LoadFromFile method for a TBLOBField component named Table1Bitmap (a field called Bitmap in the table associated with a TTable component named Table1):

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1Bitmap.LoadFromFile(
    'c:\delphi\images\splash\16color\construc.bmp');
end;
```

The third method uses the Assign method to copy the contents of an object of type TBitmap into the Picture property of a TDBImage component. An object of type TBitmap might be the Bitmap property of the Picture object property of a TImage component or it may be a stand-alone TBitmap object. As with the method copying the data from the clipboard into a TDBImage component, the bitmap data in the TDBImage component is saved into the BLOB field in the table when the record is successfully posted.

Here is an example using the Assign method. In this case, a stand-alone TBitmap object is used as the source of the bitmap data. To put a bitmap image into the TBitmap, the LoadFromFile method of the TBitmap component is called.

```
procedure TForm1.Button3Click(Sender: TObject);
var
  B: TBitmap;
begin
  B := TBitmap.Create;
  try
    B.LoadFromFile('c:\delphi\images\splash\16color\athena.bmp');
    DBImage1.Picture.Assign(B);
  finally
    B.Free;
  end;
end;
```

Going the opposite direction -- extracting a bitmap from an InterBase BLOB field (without first saving the bitmap out to a file) is a simple process of using the Assign method of the TBLOBField object to store the contents of the BLOB field to an object of type TBitmap. A stand-alone TBitmap object or the Bitmap property of the Picture object property of a TIMage component are examples of compatible destinations for this operation.

Here is an example demonstrating using the Assign method to copy a bitmap from a BLOB field into a TImage component (Table1Bitmap is the TBLOBfield for the BLOB field in the table).

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    Image1.Picture.Bitmap.Assign(Table1Bitmap);
  end;
```

In this example, the TBLOBField object Table1Bitmap is a BLOB field in an InterBase table. This TBLOBField object was created using the Fields Editor. If the Fields Editor is not used to create TFields for the fields in the table, the fields must be referenced using either the FieldByName method or the Fields property, both part of the TTable and TQuery components. In cases where one of those means is used to reference the BLOB field in a table, the field reference must be type-cast as a TBLOBField object prior to using the Assign method. For example:

```
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Image1.Picture.Bitmap.Assign(TBLOBField(Table1.Fields[1]));
  end;
```

A bitmap stored in a BLOB field may also be copied directly to a stand-alone TBitmap object. Here is an example showing the creation of a TBitmap object and storing into it a bitmap from a BLOB field.

```
  procedure TForm1.Button2Click(Sender: TObject);
  var
    B: TBitmap;
  begin
    B := TBitmap.Create;
    try
      B.Assign(Table1Bitmap);
      Image1.Picture.Bitmap.Assign(B);
    finally
      B.Free;
    end;
  en
```

Control Font Styles

NUMBER : 2812
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Control Font Styles

Control Font Styles:

This code will change the font style of a Edit
when selected. This code could be implemented to
control font style on other objects.

With a Edit(Edit1) and a ListBox(ListBox1) on a form
Add the following Items to the ListBox:
fsBold
fsItalic
fsUnderLine
fsStrikeOut

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
  X : Integer;
type
  TLookUpRec = record
    Name: String;
    Data: TFontStyle;
  end;
const
  LookUpTable: array[1..4] of TLookUpRec =
  ((Name: 'fsBold'; Data: fsBold),
   (Name: 'fsItalic'; Data: fsItalic),
   (Name: 'fsUnderline'; Data: fsUnderline),
   (Name: 'fsStrikeOut'; Data: fsStrikeOut));
begin
  X := ListBox1.ItemIndex;
  Edit1.Text := ListBox1.Items[X];
  Edit1.Font.Style := [LookUpTable[ListBox1.ItemIndex+1].Data];
end;
```

Handling EDBEngineError Exceptions

NUMBER : 2814
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Handling EDBEngineError Exceptions

Information that describes the conditions of a database engine error can be obtained for use by an application through the use of an EDBEngineError exception. EDBEngineError exceptions are handled in an application through the use of a try..except construct. When an EDBEngineError exception occurs, a EDBEngineError object would be created and various fields in that EDBEngineError object would be used to programmatically determine what went wrong and thus what needs to be done to correct the situation. Also, more than one error message may be generated for a given exception. This requires iterating through the multiple error messages to get needed information.

The fields that are most pertinent to this context are:

ErrorCount: type Integer; indicates the number of errors that are in the Errors property; counting begins at zero.

Errors: type TDBError; a set of record-like structures that contain information about each specific error generated; each record is accessed via an index number of type Integer.

Errors.ErrorCode: type DBIResult; indicating the BDE error code for the error in the current Errors record.

Errors.Category: type Byte; category of the error referenced by the ErrorCode field.

Errors.SubCode: type Byte; subcode for the value of ErrorCode.

Errors.NativeError: type LongInt; remote error code returned from the server; if zero, the error is not a server error; SQL statement return codes appear in this field.

Errors.Message: type TMessageStr; if the error is a server error, the server message for the error in the current Errors record; if not a server error, a BDE error message.

In a try..except construct, the EDBEngineError object is created directly in the except section of the construct. Once created, fields may be accessed normally, or the object may be passed to another procedure for inspection of the errors. Passing the EDBEngineError object to a specialized procedure is preferred for an application to make the process more modular, reducing the amount of repeated code for parsing the object for error information. Alternately, a custom component could be created to serve this purpose, providing a set of functionality that is easily transported across applications. The example below only demonstrates creating

the DBEngineError object, passing it to a procedure, and parsing the object to extract error information.

In a try..except construct, the DBEngineError can be created with syntax such as that below:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  if Edit1.Text > ' ' then begin
    Table1.FieldByName('Number').AsInteger := StrToInt(Edit1.Text);
    try
      Table1.Post;
    except on E: EDBEngineError do
      ShowError(E);
    end;
  end;
end;
```

In this procedure, an attempt is made to change the value of a field in a table and then call the Post method of the corresponding TTable component. Only the attempt to post the change is being trapped in the try..except construct. If an EDBEngineError occurs, the except section of the construct is executed, which creates the EDBEngineError object (E) and then passes it to the procedure ShowError. Note that only an EDBEngineError exception is being accounted for in this construct. In a real-world situation, this would likely be accompanied by checking for other types of exceptions.

The procedure ShowError takes the EDBEngineError, passed as a parameter, and queries the object for contained errors. In this example, information about the errors are displayed in a TMemo component. Alternately, the extracted values may never be displayed, but instead used as the basis for logic branching so the application can react to the errors. The first step in doing this is to establish the number of errors that actually occurred. This is the purpose of the ErrorCount property. This property supplies a value of type Integer that may be used to build a for loop to iterate through the errors contained in the object. Once the number of errors actually contained in the object is known, a loop can be used to visit each existing error (each represented by an Errors property record) and extract information about each error to be inserted into the TMemo component.

```
procedure TForm1.ShowError(AExc: EDBEngineError);
var
  i: Integer;
begin
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Number of errors: ' + IntToStr(AExc.ErrorCount));
  Memo1.Lines.Add('');
  {Iterate through the Errors records}
  for i := 0 to AExc.ErrorCount - 1 do begin
    Memo1.Lines.Add('Message: ' + AExc.Errors[i].Message);
    Memo1.Lines.Add('    Category: ' +
      IntToStr(AExc.Errors[i].Category));
```

```
      Memo1.Lines.Add('     Error Code: ' +
        IntToStr(AExc.Errors[i].ErrorCode));
      Memo1.Lines.Add('     SubCode: ' +
        IntToStr(AExc.Errors[i].SubCode));
      Memo1.Lines.Add('     Native Error: ' +
        IntToStr(AExc.Errors[i].NativeError));
      Memo1.Lines.Add('');
    end;
  end;
```

How to handle text drawing in a TDBGrid

NUMBER : 2815
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to handle text drawing in a TDBGrid

The following method can be used as the event handler for a TDBGrid.OnDrawDataCell event. This method demonstrates how to paint the text in one column a different color than the rest of the text in the grid.

```
procedure TForm1.DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect;
  Field: TField; State: TGridDrawState);
{ NOTE: DefaultDrawing propery of Grid(s) must be set to False }
begin
  { if field name is "NAME" }
  if Field.FieldName = 'NAME' then
    { change font color to red }
    (Sender as TDBGrid).Canvas.Font.Color := clRed;
  { draw text in the grid }
  (Sender as TDBGrid).Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2,
                                      Field.AsString);
end;
```

Trouble running Delphi programs from Delphi

NUMBER : 2816
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Trouble running Delphi programs from Delphi

Delphi Integrated Debugger

Delphi's integrated debugger is a feature of the IDE that allows you to execute, step, trace, and otherwise debug your Delphi applications. By default, Delphi's integrated debugger is enabled (via Options|Project|Integrated Debugging) on startup of Delphi.

The debugger handles certain interrupts and operating system debugger hooks in an effort to trap General Protection Faults and other types of exceptions. Other programs that attempt to handle exceptions in the same way will quite possibly conflict with the integrated debugger. DataSafe is one example of a program that conficts with the integrated debugger. Classic symptoms include, Delphi not recognizing that you've stopped your program and getting kicked into DOS without warning at various stages.

To test for a debugger conflict, compiling the program from Delphi, and run it from File Manager. If it runs, then disable the integrated debugger, and run your application. If it still runs, then the problem is most likely a conflict between the exception-handling software and the integrated debugger.

There are two ways to fix this. The first is to remove the conflicting software. The other is to disable the integrated debugger by un-selecting Options|Enviroment|Integrated Debugging.

Understanding the PARADOX.NET file with the BDE

NUMBER : 2817
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Understanding the PARADOX.NET file with the BDE

When using Paradox type table with the Borland Database Engine (BDE), a Paradox net file is used in the same way that Paradox does--to keep track of the number of users. Each Paradox and BDE user is given one count. So, for example, if three Paradox and three BDE users use the same Paradox net file, the user count would be six (one for each Paradox user, and one for each BDE user).

The net file also regulates access to tables. Table access is enforced through the use of lock files, which are written out to the directories containing Paradox tables. A lock file points to a particular net file, which has exclusive control over the table. This means that any user wanting access to the table must use the net file that controls the table.

If you are using Paradox for Windows 5.0 or greater, IDAPI is used for both Paradox and BDE. The "NET DIR" path is set in the IDAPICFG.EXE or BDECFG.EXE: BDE/IDAPI Configuration Utility. Once the configuration tool is running, hi-light PARADOX, which is in the "Driver Name" List Box on the "Drivers" page. Set the "NET DIR" path to a network drive where all users that are accessing the tables have write access to.

If you are using an older version of Paradox for Windows, Paradox for DOS, or the Paradox Engine, use the appropriate method for each of these products to set the netfile path to the identical path that is set within the IDAPI/BDE Configuration Utility. In the case of two BDE programs sharing tables, the net paths must be identical.

The message "Multiple Paradox net file found" (Error 0x2C06 or 11270: DBIERR_NETMULTIPLE) indicates that these rules have not been followed. If you are certain that all current users have the same net path, this message usually indicates that an old lock file exists that points to a different net file. Old lock files can be deleted if care is taken to ensure that no one is currently using them. Deleting active lock files can produce unpredictable results and could cause loss of data.

you received with the Borland product to which this information pertains.

Printing the contents of a TMemo or TListbox

NUMBER : 2809
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Printing the contents of a TMemo or TListbox

Q: How can I print all of the lines within a TMemo or TListbox component?

A: The function below accepts a TStrings object as a parameter and prints out each string to the default printer. Because it uses a TStrings, the function will work with any type of component that contains a TStrings-type property, such as a TDBMemo or TOutline.

{ Begin code listing }

```
uses Printers;

procedure PrintStrings(Strings: TStrings);
var
  Prn: TextFile;
  i: word;
begin
  AssignPrn(Prn);
  try
    Rewrite(Prn);
    try
      for i := 0 to Strings.Count - 1 do
        writeln(Prn, Strings.Strings[i]);
    finally
      CloseFile(Prn);
    end;
  except
    on EInOutError do
      MessageDlg('Error Printing text.', mtError, [mbOk], 0);
  end;
end;
```

{ End code listing }

To print out the contents of a TMemo or TListbox, use the following code:

```
PrintStrings(Memo1.Lines);
```

or

```
PrintStrings(Listbox1.Items);
```

you received with the Borland product to which this information pertains.

Searching Through Query Result Sets

NUMBER : 2820
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Searching Through Query Result Sets

The TQuery component does not offer the index-based search capabilities of the TTable component (FindKey, GotoKey, and GotoNearest). So how do you search within the result data set from a TQuery to find a row with a specific field value?

One way to search a query result set is a sequential search. This type of search starts at the first row in the data set and, in a While loop, sequentially compares the value of a field in the row with a search value. One of two results are possible: a value will be found (success) or the end of the data set will be reached (failure). The problem with this way of searching the data set is that the further into the data set a row with a matching value is, the longer it takes to arrive at that row. And, a failed search takes longest of all because it must go all the way to the last row in the data set. If the data set being searched is a large one, this process may take a considerable amount of time.

Here is a function that will perform a sequential search of the result set from a TQuery:

```
function SeqSearch(AQuery: TQuery; AField, AValue: String): Boolean;
begin
  with AQuery do begin
    First;
    while (not Eof) and (not (FieldByName(AField).AsString = AValue)) do
      Next;
    SeqSearch := not Eof;
  end;
end;
```

This function takes three parameters:

1. AQuery: type TQuery; the TQuery component in which the search is to be executed.
2. AField: type String; the name of the field against which the search value will be compared.
3. AValue: type String; the value being searched for. If the field is of a data type other than String, this search value should be changed to the same data type.

The Boolean return value of this function indicates the success (True) or failure (False) of the search.

An alternative is using a bracketing approach. On a conceptual level, this method acts somewhat like a b-tree index. It is based on the given that for a row at a given point in the data set, the value of the field being

searched compared to the search value will produce one of three possible conditions:

1. The field value will be greater than the search value, or...
2. The field value will be less than the search value, or...
3. The field value will be equal to the search value.

A bracketing search process uses this means of looking at the current row in respect to the search value and uses it to successively reduce the rows to be search by half, until only one row remains. This search field value for this sole remaining row will either be a match to the search value (success) or it will not (failure, and no match exists in the data set).

Functionally, this process lumps the condition of the search field being less than or equal to the search value into a single condition. This leaves only two possible results for the comparison of the current search field value with the search value: less than/equal to or greater than. Initially, a range of numbers is established. The low end of the range is represented by an Integer, at the start of the search process set to 0 or one less than the first row in the data set. The far end of the range is also an Integer, with the value of the RecordCount property of the TQuery. The current row pointer is then moved to a point half way between the low and high ends of the range. The search field value at that row is then compared to the search value. If the field value is less than or equal to the search value, the row being sought must be in the lower half of the range of rows so the high end of the range is reduced to the current row position. If the field value is greater than the search value, the sought value must be in the higher half of the range and so the low end is raised to the current point. By repeating this process, the number of rows that are encompassed in the range are successively reduced by half. Eventually, only one row will remain.

Putting this into a modular, transportable function, the code would look like that below:

```
function Locate(AQuery: TQuery; AField, AValue: String): Boolean;
var
  Hi, Lo: Integer;
begin
  with AQuery do begin
    First;
    {Set high end of range of rows}
    Hi := RecordCount;
    {Set low end of range of rows}
    Lo := 0;
    {Move to point half way between high and low ends of range}
    MoveBy(RecordCount div 2);
    while (Hi - Lo) > 1 do begin
      {Search field greater than search value, value in first half}
      if (FieldByName(AField).AsString > AValue) then begin
        {Lower high end of range by half of total range}
        Hi := Hi - ((Hi - Lo) div 2);
        MoveBy(((Hi - Lo) div 2) * -1);
      end
      {Search field less than search value, value in far half}
      else begin
```

```
            {Raise low end of range by half of total range}
            Lo := Lo + ((Hi - Lo) div 2);
            MoveBy((Hi - Lo) div 2);
          end;
        end;
        {Fudge for odd numbered rows}
        if (FieldByName(AField).AsString > AValue) then Prior;
        Locate := (FieldByName(AField).AsString = AValue)
      end;
    end;
```

Because there will never be a difference of less than one between the low and high ends of the range of rows, a final fudge was added to allow the search to find the search value in odd numbered rows.

This function takes the same three three parameters as the SeqSearch function described earlier.

The return value of this function is of type Boolean, and reflects the success or failure of the search. As the search does move the row pointer, the effects of this movement on the implicit posting of changed data and on where the desired position of the row pointer should be after a failed search should be taken into account in the calling application. For instance, a TBookmark pointer might be used to return the row pointer to where it was prior to a search if that search fails.

How is this process better than a sequential search? First, in bracketing the search value, only a fraction of the number of rows will be visited as would be the case in a sequential search. Unless the row with the value being sought is in the first 1,000 rows, this search method will be faster than a sequential search. Because this process always uses the same number of records, the search time will be consistent whether searching for the value in row 1,000 or row 90,000. This is in contrast with the sequential search that takes longer the farther into the data set the desired row is.

Can this method be used with any TQuery result set? No. Because of the way this method works in basing the direction of the search as either high or low, it depends on the row being ordered in a descending manner based on the field in which the search will be conducted. This means that it can only be used if the data set is naturally in a sequential order or an ORDER BY clause is used in the SQL statement for the TQuery. The size of the result set will also be a factor when deciding whether to perform a sequential or bracketing search. This process is most advantageous for speed when used with larger result sets. With smaller sets (1,00 or less rows), though, a sequential search will often be as fast or faster.

dBASE .DBF File Structure

NUMBER : 2821
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : dBASE .DBF File Structure

Sometimes it is necessary to delve into a dBASE table outside the control of the Borland Database Engine (BDE). For instance, if the .DBT file (that contains memo data) for a given table is irretrievably lost, the file will not be usable because the byte in the file header indicates that there should be a corresponding memo file. This necessitates toggling this byte to indicate no such accompanying memo file. Or, you may just want to write your own data access routine.

Below are the file structures for dBASE table files. Represented are the file structures as used for various versions of dBASE: dBASE III PLUS 1.1, dBASE IV 2.0, dBASE 5.0 for DOS, and dBASE 5.0 for Windows.

****
The data file header structure for dBASE III PLUS table file.
****

The table file header:
==========

Byte	Contents	Description
0	1 byte	Valid dBASE III PLUS table file (03h without a memo (.DBT file; 83h with a memo).
1-3	3 bytes	Date of last update; in YYMMDD format.
4-7	32-bit number	Number of records in the table.
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.
12-14	3 bytes	Reserved bytes.
15-27	13 bytes	Reserved for dBASE III PLUS on a LAN.
28-31	4 bytes	Reserved bytes.
32-n	32 bytes each	Field descriptor array (the structure of this array is shown below)
n+1	1 byte	0Dh stored as the field terminator.

n above is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table Field Descriptor Bytes
==========

Byte	Contents	Description

0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (C, D, L, M, or N).
12-15	4 bytes	Field data address (address is set in memory; not useful on disk).
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved for dBASE III PLUS on a LAN.
20	1 byte	Work area ID.
21-22	2 bytes	Reserved for dBASE III PLUS on a LAN.
23	1 byte	SET FIELDS flag.
24-31	1 byte	Reserved bytes.

Table Records
==========

The records follow the header in the table file. Data records are preceded by one byte, that is, a space (20h) if the record is not deleted, an asterisk (2Ah) if the record is deleted. Fields are packed into records without field separators orrecord terminators. The end of the file is marked by a single byte, with the end-of-file marker, an OEM code page character value of 26 (1Ah). You can input OEM code page data as indicated below.

Allowable Input for dBASE Data Types
==========

Data Type	Data Input
C (Character)	All OEM code page characters.
D (Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
N (Numeric)	- . 0 1 2 3 4 5 6 7 8 9
L (Logical)	? Y y N n T t F f (? when not initialized).
M (Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Binary, Memo, and OLE Fields And .DBT Files
==========

Memo fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). The size of these blocks are internally set to 512 bytes. The first block in the .DBT file, block 0, is the .DBT file header.

Memo field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

This information is from the Using dBASE III PLUS manual, Appendix C.

****

The data file header structure for dBASE IV 2.0 table file.
****

File Structure:
==========

Byte	Contents	Meaning
0	1byte	Valid dBASE IV file; bits 0-2 indicate version number, bit 3 the presence of a dBASE IV memo file, bits 4-6 the presence of an SQL table, bit 7 the presence of any memo file (either dBASE III PLUS or dBASE IV).
1-3	3 bytes	Date of last update; formattted as YYMMDD.
4-7	32-bit number	Number of records in the file.
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.
12-13	2 bytes	Reserved; fill with 0.
14	1 byte	Flag indicating incomplete transaction.
15	1 byte	Encryption flag.
16-27	12 bytes	Reserved for dBASE IV in a multi-user environment.
28	1 bytes	Production MDX file flag; 01H if there is an MDX, 00H if not.
29	1 byte	Language driver ID.
30-31	2 bytes	Reserved; fill with 0.
32-n*	32 bytes each	Field descriptor array (see below).
n + 1	1 byte	0DH as the field terminator.

* n is the last byte in the field descriptor array. The size of the array depends on the number of fields in the database file.

The field descriptor array:
==========

Byte	Contents	Meaning
0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (C, D, F, L, M, or N).
12-15	4 bytes	Reserved.
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved.
20	1 byte	Work area ID.
21-30	10 bytes	Reserved.
31	1 byte	Production MDX field flag; 01H if field has an index tag in the production MDX file, 00H if not.

Database records:
==========

The records follow the header in the database file. Data records are preceded by one byte; that is, a space (20H) if the record is not deleted, an asterisk (2AH) if the record is deleted. Fields are packed into records without field separators or record terminators. The end of the file is marked by a single byte, with the end-of-file marker an ASCII 26 (1AH) character.

Allowable Input for dBASE Data Types:
==========

Data	Type	Data Input
C	(Character)	All OEM code page characters.
D	(Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
F	(Floating point binary numeric)	- . 0 1 2 3 4 5 6 7 8 9
N	(Binary coded decimal numeric)	- . 0 1 2 3 4 5 6 7 8 9
L	(Logical)	? Y y N n T t F f (? when not initialized).
M	(Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Memo Fields And .DBT Files
==========

Memo fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each memo field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

This information is from the dBASE IV Language Reference manual, Appendix D.

****
The data file header structure for dBASE 5.0 for DOS table file.
****

The table file header:
==========

Byte	Contents	Description
0	1 byte	Valid dBASE for Windows table file; bits 0-2 indicate version number; bit 3 indicates presence of a dBASE IV or dBASE for Windows memo file; bits 4-6 indicate the presence of a dBASE IV SQL table; bit 7 indicates the presence of any .DBT memo file (either a dBASE III PLUS type or a dBASE IV or dBASE for Windows memo file).
1-3	3 bytes	Date of last update; in YYMMDD format.
4-7	32-bit	Number of records in the table.

	number	
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.
12-13	2 bytes	Reserved; filled with zeros.
14	1 byte	Flag indicating incomplete dBASE transaction.
15	1 byte	Encryption flag.
16-27	12 bytes	Reserved for multi-user processing.
28	1 byte	Production MDX flag; 01h stored in this byte if a production .MDX file exists for this table; 00h if no .MDX file exists.
29	1 byte	Language driver ID.
30-31	2 bytes	Reserved; filled with zeros.
32-n	32 bytes each	Field descriptor array (the structure of this array is shown below)
n+1	1 byte	0Dh stored as the field terminator.

n above is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table Field Descriptor Bytes
==========

Byte	Contents	Description
0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (B, C, D, F, G, L, M, or N).
12-15	4 bytes	Reserved.
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved.
20	1 byte	Work area ID.
21-30	10 bytes	Reserved.
31	1 byte	Production .MDX field flag; 01h if field has an index tag in the production .MDX file; 00h if the field is not indexed.

Table Records
==========

The records follow the header in the table file. Data records are preceded by one byte, that is, a space (20h) if the record is not deleted, an asterisk (2Ah) if the record is deleted. Fields are packed into records without field separators orrecord terminators. The end of the file is marked by a single byte, with the end-of-file marker, an OEM code page character value of 26 (1Ah). You can input OEM code page data as indicated below.

Allowable Input for dBASE Data Types
==========

Data Type	Data Input
C (Character)	All OEM code page characters.
D (Date)	Numbers and a character to separate month, day, and year

(stored internally as 8 digits in YYYYMMDD format).

F (Floating point binary numeric)	- . 0 1 2 3 4 5 6 7 8 9
N (Numeric)	- . 0 1 2 3 4 5 6 7 8 9
L (Logical)	? Y y N n T t F f (? when not initialized).
M (Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Memo Fields And .DBT Files
==========

Memo fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each memo field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

Unlike dBASE III PLUS, if you delete text in a memo field, dBASE 5.0 for DOS may reuse the space from the deleted text when you input new text. dBASE III PLUS always appends new text to the end of the .DBT file. In dBASE III PLUS, the .DBT file size grows whenever new text is added, even if other text in the file is deleted.

This information is from the dBASE for DOS Language Reference manual, Appendix C.

****
The data file header structure for dBASE 5.0 for Windows table file.
****

The table file header:
==========

Byte	Contents	Description
0	1 byte	Valid dBASE for Windows table file; bits 0-2 indicate version number; bit 3 indicates presence of a dBASE IV or dBASE for Windows memo file; bits 4-6 indicate the presence of a dBASE IV SQL table; bit 7 indicates the presence of any .DBT memo file (either a dBASE III PLUS type or a dBASE IV or dBASE for Windows memo file).
1-3	3 bytes	Date of last update; in YYMMDD format.
4-7	32-bit number	Number of records in the table.
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.

12-13	2 bytes	Reserved; filled with zeros.
14	1 byte	Flag indicating incomplete dBASE IV transaction.
15	1 byte	dBASE IV encryption flag.
16-27	12 bytes	Reserved for multi-user processing.
28	1 byte	Production MDX flag; 01h stored in this byte if a production .MDX file exists for this table; 00h if no .MDX file exists.
29	1 byte	Language driver ID.
30-31	2 bytes	Reserved; filled with zeros.
32-n	32 bytes each	Field descriptor array (the structure of this array is shown below)
n+1	1 byte	0Dh stored as the field terminator.

n above is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table Field Descriptor Bytes
==========

Byte	Contents	Description
0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (B, C, D, F, G, L, M, or N).
12-15	4 bytes	Reserved.
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved.
20	1 byte	Work area ID.
21-30	10 bytes	Reserved.
31	1 byte	Production .MDX field flag; 01h if field has an index tag in the production .MDX file; 00h if the field is not indexed.

Table Records
==========

The records follow the header in the table file. Data records are preceded by one byte, that is, a space (20h) if the record is not deleted, an asterisk (2Ah) if the record is deleted. Fields are packed into records without field separators orrecord terminators. The end of the file is marked by a single byte, with the end-of-file marker, an OEM code page character value of 26 (1Ah). You can input OEM code page data as indicated below.

Allowable Input for dBASE Data Types
==========

Data Type	Data Input
B (Binary)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).
C (Character)	All OEM code page characters.
D (Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
G (General	All OEM code page characters (stored internally as 10 digits or OLE) representing a .DBT block number).

N (Numeric) - . 0 1 2 3 4 5 6 7 8 9
L (Logical) ? Y y N n T t F f (? when not initialized).
M (Memo) All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Binary, Memo, and OLE Fields And .DBT Files

Binary, memo, and OLE fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each binary, memo, or OLE field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

Unlike dBASE III PLUS, if you delete text in a memo field (or binary and OLE fields), dBASE for Windows (unlike dBASE IV) may reuse the space from the deleted text when you input new text. dBASE III PLUS always appends new text to the end of the .DBT file. In dBASE III PLUS, the .DBT file size grows whenever new text is added, even if other text in the file is deleted.

This information is from the dBASE for Windows Language Reference manual, Appendix C.

Setting File Handles For A Windows BDE Application

NUMBER : 2822
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Setting File Handles For A Windows BDE Application

"Setting the number of file handles for Borland Database Engine Applications"

Solving the Borland Database Engine (BDE) error, "Not Enough file handles" (Error 0x2502 or 9474: DBIERR_NOFILEHANDLES), is not a difficult task, but it requires an understanding of how file handles are used by DOS, Windows, and your application.

At the lowest level, the number of file handles available to any application running on your computer is specified in the "FILES=nn" statement of CONFIG.SYS. For Windows running with Windows and DOS applications, a bare minimum value is 30-40. A value of 80-120 is a more realistic starting value, which you should increase according to your application's needs.

When your application starts in Windows, the number of files available to it is based on the following two values: (1) The number you specify in a call to the Windows API call 'SetHandleCount' (default = 20), and (2) the number of files specified in the _NFILE_ constant in Borland's RTL source (default = 20). These two values should be the same; if they're not, the lower one will apply.

If your application requires more than the default twenty file handles for its own use, you'll need to follow the procedures outlined in Borland's TI-870. This document is available on our Download BBS and Compuserve as TI870.ZIP and on TechFax as document #870. When estimating the number you'll need, note that in addition to whatever files you need for the BDE and for files you want to open, a C/C++ application will open five file handles for stdin, stdout, etc., on startup.

The number of file handles the BDE itself can use is specified on the System Page in the MaxFileHandles parameter of the BDE Configuration utility. It is important to realize that the number specified by MaxFileHandles is the number of file handles that the BDE will expect to receive for its own exclusive use, not for use by your application as a whole. The BDE doesn't allocate file handles on its own, it gets these from your application. Thus, if you make a call to SetHandleCount with a value of 20 and set the MaxFileHandles parameter to 20, all twenty handles may go to the BDE, leaving none for your application's other needs.

To sum up, the number of file handles your BDE application

will need will be the sum of 1) five standard file handles, 2) the number needed by the BDE (MaxFileHandles), and 3) the number you need for other file manipulation routines. This sum should be the number you specify in SetHandleCount, and if this total is greater than 20, you'll need to change the RTL source according to TI870 as well.

Passing a Variable to a ReportSmith Report

NUMBER : 2824
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Passing a Variable to a ReportSmith Report

The following code demonstrates how to pass variables to a ReportSmith report.

Note: this technical information sheet assumes that you know how to create a report that includes report variables. For additional information see chapter four of the ReportSmith manual 'Creating Reports' that shipped with Delphi.

In this case we are assigning a value ('CA') to a string report variable named 'state'.

ReportSmith Code:
This is the info in the ReportVariables Dialog box in ReportSmith. You can get to this dialog from the reportsmith menu by choosing Tools | Report Query and pushing the Report Variables button.

Name: state
Type: string
Title: state var
Prompt: Enter your favorite state.
Entry: type-in
Report Variables: state ; Note this variable and the value it holds are both case sensitive when passed to ReportSmith.

Delphi Code:

This code assumes that you have placed a TReport component on your form named 'Report1' and set the ReportName property to the name of the report that will be accepting the variable as defined above.

Place the following code in the OnClick Method of a pushbutton on your form. I use a pushbutton for simplicity, but this code could just as easily be triggered by any other event.

```
procedure TForm1.Button1Click(Sender: TObject);
  var s: string;
  begin
    s:='CA';
    Report1.InitialValues.Add('@state=<'+s+'>');
    Report1.run;
  end;
```

Extracting Index Data From A Table

NUMBER : 2825
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Extracting Index Data From A Table

Getting a list of the indexes associated with a table at run-time can be as simple as a call to the GetIndexNames method of the TTable, TQuery, or TStoredProc component. The GetIndexNames method returns a list of the that are available for the data set in the form of a TStrings list, which may then be inserted into such visual components as a TListBox through its Items property:

```
ListBox1.Clear;
Table1.GetIndexNames(ListBox1.Items);
```

Of course, the TStrings list returned by the GetIndexNames method need not be used with a visual component. It could just as well serve as an array of index names stored entirely in memory, that can be used as a list or array.

But it is also possible to retrieve much more information about the indexes for a table than just the names. Other descriptive attributes include the name of each index, the names of the fields that comprise each index, and the index options used when each index was created. Retrieving these values is slightly more involved than the use of the GetIndexNames. Basically, this process involves iterating through the IndexDefs property of the TTable, TQuery, or TStoredProc component. The IndexDefs property is essentially an array of records, one record for each index for the table. Each index record contains information about the index. It is a relatively straightforward process to iterate through this array of index descriptions, extracting information about individual indexes.

The IndexDefs property of the TTable component contains information about the indexes for the table used by the TTable, TQuery, or TStoredProc component in use. The IndexDefs property itself has various properties that allow for the extraction of information about specific indexes. The two properties in the IndexDefs object are:

Count: type Integer; available only at run-time and read-only; indicates the number ofentries in the Items property (i.e., the number of indexes in the table).
Items: type TIndexDef; available only at run-time and read-only; an array of TIndexDef objects, one for each index in the table.

The Count property of the IndexDefs object is used as the basis for a loop program construct to iterate through the Items property entries to extract specific information about each index. Each IndexDef object contained in the Items property consists of a number of properties that provide various bits of information that describe each index. All of the properties of the IndexDef object are available only at run-time and are

all read-only. These properties are:

Expression: type String; indicates the expression used for dBASE multi-field indexes.
Fields: type String; indicates the field or fields upon which the index is based.
Name: type String; name of the index.
Options: type TIndexOptions; characteristics of the index (ixPrimary, ixUnique, etc.).

Before any index information (Count or Items) can be accessed, the Update method of the IndexDefs object must be called. This refreshes or initializes the IndexDef object's view of the set of indexes.

Examples
========

Here is a simple For loop based on the Count property of the IndexDefs object that extracts the name of each index (if any exist) for the table represented by the TTable component Table1:

```
procedure TForm1.ListBtnClick(Sender: TObject);
var
  i: Integer;
begin
  ListBox1.Items.Clear;
  with Table1 do begin
    if IndexDefs.Count > 0 then begin
      for i := 0 to IndexDefs.Count - 1 do
        ListBox1.Items.Add(IndexDefs.Items[i].Name)
    end;
  end;
end;
```

Below is an example showing how to extract information about indexes at run-time, plugging the extracted values into a TStringGrid (named SG1).

```
procedure TForm1.FormShow(Sender: TObject);
var
  i: Integer;
  S: String;
begin
  with Table1 do begin
    Open;
    {Refresh IndexDefs object}
    IndexDefs.Update;
    if IndexDefs.Count > 0 then begin
      {Set up columns and rows in grid to match IndexDefs items}
      SG1.ColCount := 4;
      SG1.RowCount := IndexDefs.Count + 1;
      {Set grid column labels to TIndexDef property names}
      SG1.Cells[0, 0] := 'Name';
      SG1.ColWidths[0] := 200;
      SG1.Cells[1, 0] := 'Fields';
      SG1.ColWidths[1] := 200;
      SG1.Cells[2, 0] := 'Expression';
```

```
        SG1.ColWidths[2] := 200;
        SG1.Cells[3, 0] := 'Options';
        SG1.ColWidths[3] := 300;
        {Loop through IndexDefs.Items}
        for i := 0 to IndexDefs.Count - 1 do begin
          {Fill grid cells for current row}
          SG1.Cells[0, i + 1] := IndexDefs.Items[i].Name;
          SG1.Cells[1, i + 1] := IndexDefs.Items[i].Fields;
          SG1.Cells[2, i + 1] := IndexDefs.Items[i].Expression;
          if ixPrimary in IndexDefs.Items[i].Options then
            S := 'ixPrimary, ';
          if ixUnique in IndexDefs.Items[i].Options then
            S := S + 'ixUnique, ';
          if ixDescending in IndexDefs.Items[i].Options then
            S := S + 'ixDescending, ';
          if ixCaseInsensitive in IndexDefs.Items[i].Options then
            S := S + 'ixCaseInsensitive, ';
          if ixExpression in IndexDefs.Items[i].Options then
            S := S + 'ixExpression, ';
          if S > ' ' then begin
            {Get rid of trailing ", "}
            System.Delete(S, Length(S) - 1, 2);
            SG1.Cells[3, i + 1] := S;
          end;
        end;
      end;
    end;
  end;
```

Special Considerations
======================

There are idiosyncracies associated with extracting information about indexes for different table types that Delphi can access.

dBASE Tables

With dBASE indexes, which properties of Fields and Expression will be filled will depend on the type of index, simple (single-field) or complex (based on multiple fields or a dBASE expression). If the index is a simple one, the Fields property will contain the name of the field in the table on which the index is based and the Expression property will be blank. If the index is a complex one, the Expression property will show the expression on which the index is based (e.g., "Field1+Field2") and the Fields property will be blank.

Paradox Tables

With Paradox primary indexes, the Name property will be blank, the Fields property will contain the field(s) on which the index is based, and the Options property will contain ixPrimary. With secondary indexes, the Name property will contain the name of the secondary index, the Fields property will contain the field(s) on which the index is based, and the Options property may or may not have values.

The Fields property for indexes based on more than one field will show the field names separated by semi-colons. Indexes based on only a single field will show the name of only that one field in the Fields property.

InterBase Tables

For both index types, single- or multiple-field, the Expression property will be blank. For single-field indexes, the Fields property will contain the field on which the index is based. For multi-field indexes, the Fields property will show all of the multiple fields that comprise the index, each separated by a semi-colon.

Indexes designated as PRIMARY when the CREATE TABLE command is issued will have "RDB$PRIMARYn" in the Name property, where n is a number character uniquely identifying the primary index within the database metadata. Secondary indexes will show the actual name of the index.

Foreign key constraints also result in an index being created by the system. These indexes appear in the IndexDefs property, and will have the name "RDB$FOREIGNn" where n is a number character that uniquely identifies the index within the database metadata.

The Fields property for indexes based on more than one field will show the field names separated by semi-colons. Indexes based on only a single field will show the name of only that one field in the Fields property.

Redistributing the Borland Database Engine

NUMBER : 2834
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Redistributing the Borland Database Engine

Follow the following steps in order to deploy the redistributable BDE engine on a target machine:

1) Format two diskettes in the diskette drive of the target machine. Label the diskettes "Disk 1" and "Disk 2".

2) From the DELPHI CD, Copy the \REDIST\BDEINST\DISK1 directory to the diskette labeled "Disk 1" and \REDIST\BDEINST\DISK2 to the diskette "Disk 2".

3) Insert the diskette labeled "BDE Install 1" to a floppy drive (We'll use drive A: in this example) of the target machine.

4) Make sure there are no other programs running in Windows. From the Windows Program Manager select File|Run, enter "A:\DISK1\SETUP" in the space labeled "Command Line" and press "OK" to begin install of the Borland Database Engine on the target machine.

5) A dialog labeled "Database Engine Install" will appear briefly, then a dialog labeled "preparing to install...", and finally a background screen labeled "BDE Redistributable" will appear along with a dialog allowing you to Continue or Exit. Press "Continue".

6) A dialog "Borland Database Engine Location Settings" will appear and allow you to change the path for BDE programs and configuration file. Leave the default settings at and press "Continue".

7) The "Borland Database Engine Installation" dialog will appear and allow you to display previous dialogs, or execute the install. Press "Install".

8) Installation progress will display while files from the "Disk 1" diskette are processed.

9) The "BDE Redistributable Install Request" dialog will appear. Load the diskette labeled "Disk 2". Press "continue".

10) On completion, the "Borland Database Engine Installation Notification" dialog will inform you that BDE is installed. Press "Exit".

11) Exit Windows, remove any floppy disks from your drives and reboot the target machine.

If default settings were used, changes listed below will have taken place.

Two new directories, \IDAPI, and \IDAPI\LANGDRV will exist on the target machine. Note that the BDE Configuration Utility, BDECFG.EXE is located in the \IDAPI directory. Language drivers will now be located in \IDAPI\LANGDRV as *.LD files. AUTOEXEC.BAT, CONFIG.SYS, AND SYSTEM.INI will not be affected.

WIN.INI in the \WINDOWS\SYSTEM directory will have new entries:

```
[IDAPI]
DLLPATH=C:\IDAPI
CONFIGFILE01=C:\IDAPI\IDAPI.CFG

[Borland Language Drivers]
LDPath=C:\IDAPI\LANGDRV
```

Cascading Deletes With Pdox Referential Integrity

NUMBER : 2837
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Cascading Deletes With Pdox Referential Integrity

Paradox tables offer a Referential Integrity feature. This feature prevents adding records to a child table for which there is no matching record in the parent table. It will also cause the key field(s) in the child table to be changed when the corresponding key field(s) in the parent are changed (commonly referred to as a cascading update). These events occur automatically, requiring no intervention by a Delphi application using these tables. However, the Paradox Referential Integrity feature will not accommodate cascading deletes. That is, Delphi will not allow you to delete a record in the parent table while matching records exist in the child table. This would make "orphans" of the child records, losing referential integrity. Delphi raises an exception when an attempt is made to delete such a parent record.

To effect a cascading delete requires that the deletion of the matching child records be deleted programmatically -- before the parent record is deleted. In a Delphi application, this is done by interrupting the process of deleting the record in the parent table, deleting the matching records in the child table (if there are any), and then continuing with the deletion of the parent record.

A record in a table is deleted by a call to the Delete method of the TTable component, which deletes the current record in the associated table. Interrupting the this process to first perform some other operations is a matter creating a procedure associated with the BeforeDelete event of the TTable. Any commands in a BeforeDelete event procedure are executed before the call actually goes out from the application to the Borland Database Engine (BDE) to physically remove the record from the table file.

To handle the deletion of one or more child records, in a BeforeDelete event procedure the Delete method for the TTable representing the child table is called in a loop. The loop is based on the condition of the record pointer in the table not being positioned at the end of the data set, as indicated by the Eof method of the TTable. This also accounts for there being no child records at all matching the parent record to be deleted: if there are no matching records, the record pointer will already be at the end of the data set, the loop condition will evaluate to False, and the Delete method in the loop nevers gets executed.

```
procedure TForm1.Table1BeforeDelete(DataSet: TDataset);
begin
  with Table2 do begin
    DisableControls;
    First;
    while not Eof do
```

```
        Delete;
      EnableControls;
    end;
  end;
```

In the above example, the parent table is represented by the TTable component Table1 and the child by Table2. The DisableControls and EnableControls methods are used as a cosmetic measure to freeze any data-aware components that might be displaying data from Table2 while the records are being deleted. These two methods make the process visually appear smoother, but are only optional and not essential to this process. The Next method need not be called within this loop. This is because the loop begins at the first record and, as each record is deleted, the record that previously followed the deleted record moves up in the data set, becoming both the first and the current record.

This example presumes that the parent and child tables are linked with a Master-Detail relationship, as is typical for tables for which such Referntial Integrity is configured. Linking the tables in this manner results in only those records in the child table that match the current record in the parent table being available. All other records in the child table are made unavailable through the Master-Detail filtering. If the tables are not so linked, there are two additional considerations that must be accounted for when deleting the child records. The first is that a call to the First method may or may not put the record pointer on a record that matches the current record in the parent table. This necessitates using a search method to manually move the record pointer to a matching record. The second consideration affects the condition for the loop. Because records other than those matching the current record in the parent table will be accessible, the condition for the loop must check that each record is a matching record before attempting to delete it. This checking is in addition to querying the Eof method. Because the records will be ordered by this key field (from a primary or secondary index), all of the matching records will be contiguous. This leads to the given that, as soon as the first non-matching record is reached, it can be assumed that all matching records have been deleted. Thus, the previous example would be modified to:

```
  procedure TForm1.Table1BeforeDelete(DataSet: TDataset);
  begin
    with Table2 do begin
      DisableControls;
      FindKey([Table1.Fields[0].AsString])
      while (Fields[0].AsString = Table1.Fields[0].AsString) and
        (not Eof) do
        Delete;
      EnableControls;
    end;
  end;
```

In the above, it is the first field in the parent table (Table1) upon which the Referential Integrity is based, and the first field in the child table (Table2) against which matching is judged.

How to copy files in Delphi.

NUMBER : 2855
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to copy files in Delphi.

Q: How do I copy a file?

A: Here are three ways:

```
{This way uses a File stream.}
Procedure FileCopy( Const sourcefilename, targetfilename: String );
Var
  S, T: TFileStream;
Begin
  S := TFileStream.Create( sourcefilename, fmOpenRead );
  try
    T := TFileStream.Create( targetfilename,
                             fmOpenWrite or fmCreate );
    try
      T.CopyFrom(S, S.Size ) ;
    finally
      T.Free;
    end;
  finally
    S.Free;
  end;
End;

{This way uses memory blocks for read/write.}
procedure FileCopy(const FromFile, ToFile: string);
 var
  FromF, ToF: file;
  NumRead, NumWritten: Word;
  Buf: array[1..2048] of Char;
begin
  AssignFile(FromF, FromFile);
  Reset(FromF, 1);                { Record size = 1 }
  AssignFile(ToF, ToFile);        { Open output file }
  Rewrite(ToF, 1);                { Record size = 1 }
  repeat
    BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
    BlockWrite(ToF, Buf, NumRead, NumWritten);
  until (NumRead = 0) or (NumWritten <> NumRead);
  CloseFile(FromF);
  CloseFile(ToF);
end;

{This one uses LZCopy, which USES LZExpand.}
procedure CopyFile(FromFileName, ToFileName: string);
var
```

```
  FromFile, ToFile: File;
begin
  AssignFile(FromFile, FromFileName); { Assign FromFile to FromFileName }
  AssignFile(ToFile, ToFileName);     { Assign ToFile to ToFileName }
  Reset(FromFile);                    { Open file for input }
  try
    Rewrite(ToFile);                  { Create file for output }
    try
      { copy the file an if a negative value is returned }
      { raise an exception }
      if LZCopy(TFileRec(FromFile).Handle, TFileRec(ToFile).Handle) < 0
        then
        raise EInOutError.Create('Error using LZCopy')
    finally
      CloseFile(ToFile);   { Close ToFile }
    end;
  finally
    CloseFile(FromFile);   { Close FromFile }
  end;
end;
```

Removing the vertical scrollbar from a TDBGrid

NUMBER : 2840
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Removing the vertical scrollbar from a TDBGrid

In order to remove the vertical scrollbar from a TDBGrid component, you must override its Paint method. Inside the Paint method you must call the SetScrollRange API procedure to set the min and max scroll values to zero (this disables the scrollbar), and then call the inherited Paint. The code below is a unit containing a new component called TNoVertScrollDBGrid that does this. You can copy the code into a file called NEWGRID.PAS, and add it to the component library as a custom component.

```
unit Newgrid;

interface

uses
  WinTypes, WinProcs, Classes, DBGrids;

type
  TNoVertScrollDBGrid = class(TDBGrid)
  protected
    procedure Paint; override;
  end;

procedure Register;

implementation

procedure TNoVertScrollDBGrid.Paint;
begin
  SetScrollRange(Self.Handle, SB_VERT, 0, 0, False);
  inherited Paint;
end;

procedure Register;
begin
  RegisterComponents('Data Controls', [TNoVertScrollDBGrid]);
end;

end.
```

Delphi Consultants and Training Centers

NUMBER : 2841
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Delphi Consultants and Training Centers

Listings of Delphi consultants and training centers are available from Borland Fast Fax at 1-800-408-0001. From Fast Fax, select 1 for "General Information About Borland and its Products," then select 3 for "Third Party Consultants, Developers, System Integrators, and Training Centers." These lists are also available from CompuServe and the Borland Download BBS. From the Delphi CompuServe Forum, download the files DEV.ZIP and TRN.ZIP. From the Borland Download BBS, download the file DEV.ZIP and TRN.ZIP from the Delphi - General file area of the Delphi Conference. To access the Borland Download BBS, call 1-408-431-5096. For additional information about Borland Online Services, refer to Technical Information Sheet #9604.

If you would like to receive assistance from Borland's third party Connections members who offer specific software solutions for your industry, please download TI 9680 "Borland Connection Solution Request Form," fill it out, and return it back by faxing it to Borland Connections at (408) 431-4142.

InterBase BLOB Fields: A Primer

NUMBER : 2842
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : InterBase BLOB Fields: A Primer

InterBase BLOB fields are not all the same. They actually consist in a variety of forms, or sub-types of the general BLOB type. Knowing which sub-type of BLOB field to use when is essential to creating database applications that incorporate InterBase BLOB fields. BLOB fields come in three varieties: sub-type 0 and sub-type 1 (the two predefined sub-types), and user-defined sub-types.

Sub-type 0 BLOB fields are the type created when a CREATE command is issued and a sub-type is not specified. For clarity in SQL syntax, though, it is possible to explicitly indicate that the BLOB field is to be of sub-type 0. This sub-type of BLOB field is for the storage of binary data. InterBase makes no analysis of the data stored, it just stores it in the BLOB field on a byte-for-byte basis. The most common intended use for BLOB fields in Windows applications is the storage of bitmap binary data, typically for display in a TDBImage component. Either the BLOB field sub-type 0 or a user-defined sub-type BLOB field will work for this purpose.

The second predefined sub-type is 1. This BLOB field sub-type is designed for the storage of text. Typically, this is the free-form memo or notes data displayed and edited with the TDBMemo component. This BLOB field sub-type is better for storing text data than the VARCHAR field because, unlike with the VARCHAR field, there is no design-time limit placed on the storage capacity of the field.

In SQL syntax, the sub-type 1 BLOB field is created by following the BLOB field type keyword with the SUB_TYPE keyword and the integer one:

```
CREATE TABLE WITHBLOB
(
  ID CHAR(3) NOT NULL PRIMARY KEY,
  MEMO BLOB SUB_TYPE 1,
  AMOUNT NUMERIC
)
```

Aside from the two predefined BLOB field sub-types, there are user-defined sub-types. User-defined sub-types are designated by a negative integer value in association with the SUB_TYPE keyword. The actual integer value, as long as it is negative, is actually arbitrary and up to the discretion of the table creator. A designation of -1 is functionally the same as that of a -2. The only consideration when using user-defined sub-types is ensuring that the same type of binary data is stored for every row in the table for a BLOB field of a given user-defined sub-type. InterBase will not evaluate whether this criteria is met, and it is the responsibility of the application inserting the binary data to store the appropriate type of data. No error will occur from the InterBase side if an incorrect type of

binary data is stored in a user-defined BLOB field sub-type, but an application can incur difficulties if it is expecting one type of data but encounters another.

A BLOB field of a user-defined sub-type is created with the SQL syntax such as that below:

```
CREATE TABLE IMAGE_DATA
(
  FILENAME CHAR(12) NOT NULL PRIMARY KEY,
  BITMAP BLOB SUB_TYPE -1,
  EXEs BLOB SUB_TYPE -2,
)
```

When using a table created with the above command, the field BITMAP would only be used to store one distinct type of binary data for all records. In this case, bitmap data. The field EXEs implies the storage of executable files loaded from disk. If an application using this table were to mistakenly store binary data that should have been in the EXEs field into the BITMAP field, InterBase would generate no errors, but the application would have extreme difficulties displaying a stored executable file in a TDBImage component.

InterBase BLOB fields and Delphi

When defining TField objects for InterBase BLOB fields in Delphi, the various BLOB field sub-types are assigned TField derivative types as follows:

```
Sub-type 0:     TBlobField
Sub-type 1:     TMemoField
User-defined: TBlobField
```

Because both the predefined sub-type 0 and user-defined sub-types are recognized as TBlobField objects, care must be taken when designing an application to not mistake a field of one sub-type for that of another. The only way to differentiate between a field of sub-type 0 from that of a user-defined type is by viewing the metadata information for the table, which cannot be done from within Delphi. The Local InterBase Server utility WISQL can be used to view table metadata.

InterBase BLOB fields and Database Desktop

The Database Desktop utility that comes with Delphi (DBD) does not create user-defined subtypes. When using BLOB fields created in Database Desktop, use the "BLOB" field type for binary data, including bitmap data. This creates a BLOB field of the predefined sub-type 0.

The DBD also offers a BLOB field type TEXT BLOB. This is equivalent to the pre-defined subtype 1, and should be used where free-form text storage will be needed. While it is functionally equivalent to the pre-defined subtype 1 BLOB field, it will appear with a slightly different type designation if you view the metadata for the table in the WISQL utility.

Using The ASCII Driver With Comma-delimited Files

NUMBER : 2844
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Using The ASCII Driver With Comma-delimited Files

Delphi (and the BDE) has the capability to use ASCII files to a limited degree as tables. The ASCII driver has the capability to translate the data values in an ASCII fixed-length field or a comma-delimted file into fields and values that can be displayed through a TTable component. How this translation of the ASCII file takes place depends on an accompanying schema file. The schema file for an ASCII data file defines various attributes necessary for parsing the ASCII data file into individual field values. The field definitions for an ASCII fixed-length field file is relatively straightforward, the offsets of various fields in the ASCII file being consistent across all rows in the file. However, for comma-delimited files, this process is slightly more complicated due to the fact that not all data values in such a file may be the same length for all rows in the file. This article, then, concentrates on this more difficult task of reading data from comma-delimited, or varying-length field, files.

The Schema File

The schema file for an ASCII data file contains information that defines both the file type (comma-delimited versus fixed-length field), as well as defining the fields that are represented by the data values in each row of the ASCII data file. (All of the settings used in a schema file are case insensitive, so "ascii" is just as valid as "ASCII".) In order that a schema file be recognized as such, it must have the same filename as the ASCII data file for which it provides definitions, but with the filename extension .SCH (for SCHema). The attributes that describe the file are:

File name: Enclosed in square brackets, this setting specifies the name of the ASCII data file (sans the filename extension, which must be .TXT).

Filetype: Specifies whether the ASCII data file is structured as a fixed-length field file (use a setting of FIXED) or a comma-delimited file (with data values of potentially varying length (use a setting of VARYING).

Delimiter: Specifies the character that surrounds String type data values (typically, the double quotation mark, ASCII decimal 34).

Separator: Specifies the character that is used to separate individual data values (typically, a comma). This character must be a visible character, i.e., cannot be a space (ASCII decimal 32).

CharSet: Specifies the language driver (use a setting of ASCII).

Following the file definition settings are field definitions, one for each data value on each row of the ASCII data file. These field definitions supply the information Delphi and the BDE will need to create a virtual field in memory to hold the data value, that virtual field's data type which will affect how the value is translated after being read from the ASCII file, and size and positioning attributes. The various settings that will appear in each field definition are:

Field: Virtual field name, will always be "Field" followed by an integer number representing that field's ordinal position in respect to the other fields in the ASCII data file. E.G., the first field is Field1, the second Field2, and so on.

Field name: Specifies the display name for the field, which appears as the column header in a TDBGrid. Naming convention for ASCII table fields follows that for Paradox tables.

Field type: Specifies the data tyoe BDE is to use in translating the data value for each field and tells Delphi what type of virtual field to create.

Use the setting	For values of type
CHAR	Character
FLOAT	64-bit floating point
NUMBER	16-bit integer
BOOL	Boolean (T or F)
LONGINT	32-bit long integer
DATE	Date field.
TIME	Time field.
TIMESTAMP	Date + Time field.

(The actual format for date and time data values will be determined by the current setting in the BDE configuration, Date tab page.)

Data value length: Maximum length of a field's corresponding data value. This setting determines the length of the virtual field that Delphi creates to receive values read from the ASCII file.

Number of decimals: Applicable to FLOAT type fields; specifies the number of digit positions to the right of the decimal place to include in the virtual field definition.

Offset: Offset from the left that represents the starting position for the field in relation to all of the fields that preceed it.

For example, the field definition below is for the first field in the ASCII table. It defines a String type data value with a name of "Text",

a maximum data value length of three characters (and the field will appear as only three characters long in Delphi data-aware components such as the TDBGrid), no decimal places (a String data value will never have any decimal places), and an offset of zero (because it is the first field and there would not be any preceeding fields).

```
Field1=Text,Char,3,00,00
```

Here is an example of a schema file with three fields, the first of String type and the second and third of type date. This schema file would be contained in a file named DATES.SCH to provide file and field definitions for an ASCII data file named DATES.TXT.

```
[DATES]
Filetype=VARYING
Delimiter="
Separator=,
CharSet=ascii
Field1=Text,Char,3,00,00
Field2=First Contact,Date,10,00,03
Field3=Second,Date,10,00,13
```

This schema defines a comma-delimited field where all String type data values can be recognized as being surrounded by the double quotation mark and where distinct data values are separated by commas (excepting any commas that may appear within the specified delimiter, inside individual String data values). The character field has a length of three characters, no decimal places, and an offset of zero. The first date field has a length of 10, no decimals, and an offset of three. And the second date field has a length of 10, no decimals, and an offset of 13.

For reading ASCII comma-delimited files, the length and offset parameters for the field definitions do not apply to data values in the ASCII files (as is the case for fixed-length field files), but to the virtual fields, defined in the application, into which the values read will be placed. The length parameter will need to reflect the maximum length of the data value for each field -- not counting the delimiting quotation marks or the comma separators. This is most difficult to estimate for String type data values as the actual length of such a data value may vary greatly from row to row in the ASCII data file. The offset parameter for each field will not be the position of the data value in the ASCII file (as is the case for fixed-length field files), but the offset as represented by the cumulative length of all preceding fields (again, the defined fields in memory, not the data values in the ASCII file.

Here is a data file that would correspond to the schema file described above, in a file named DATES.TXT:

```
"A",08/01/1995,08/11/1995
"BB",08/02/1995,08/12/1995
"CCC",08/03/1995,08/13/1995
```

The maximum length of an actual data value in the first field is three ("CCC"). because this is the first field and there are no preceding fields, the offset for this field is zero. The length of this first field (3) is used as the offset for the second field. The length of the second

field, a date value, is 10, reflecting the maximum length of a data value for that field. The accumulated length of the first and second fields are then used as the offset for the third field (3 + 10 = 13) .

It is only when the proper length for the data values in the ASCII file are used and each fields length added to any preceding fields to produce offset values for succeeding fields that this process will correctly read the data. If data is misread because of improper length settings in the schema file, most values will suffer adverse translation effects, such as truncation of character data or numeric values being interpreted as zeros. Data will usually still be displayed, but no error should occur. However, values that must be in a specific format in order to be translated into the appropriate data type will cause errors if the value read includes characters not valid in a date value. This would include a date data value which, when incorrectly read may contain extraneous characters from other surrounding fields. Such a condition will result in a data translation exception requiring an adjustment of the field length and offset settings in the schema file.

Determining Record Number In A dBASE Table

NUMBER : 2849
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Determining Record Number In A dBASE Table

dBASE tables employ a fairly static record numbering system. The record number for a given record reflects its physical position in the table file. These record number are not subject to change dependent on the filtering of data or index ordering. For instance, a record that is the first record stored in the .DBF file would be record number 1. It is possible that, through the ordering of an index, this record may be displayed as the last of 100 records. In such a case, its record would remain the same -- one -- and would not be changed to 100 to reflect its position in the index ordered data set. This is in contrast with Paradox tables, where there is a sequence number. The Paradox sequence number is like the dBASE record number except that it is much more fluid and the number for a given record will reflect its position relative to the data set. That is, a record may not always have the same sequence number given filtering of the data set to reduce the number of records or when an index is active that may change the displayed order of the record.

In database applications created with Delphi and the Borland Database Engine (BDE), there is no provision built into the stock data components for extracting or determining the record for dBASE tables. Such an operation is, however, possible by making a call from the application a BDE function.

There are a number of BDE functions that will return information about the current dBASE record, such as the record number. Basically, any function that fills a BDE pRECProps structure would suffice. Such BDE functions include DbiGetRecord, DbiGetNextRecord, and DbiGetPriorRecord. Of course, only the first of these functions really applies to retrieving information about the current record. The other two move the record pointer when invoked, similar in effect to the Next and Prior methods of the TTable or TQuery components.

The pRECProps structure consists of the fields:

iSeqNum: type LongInt; specifies the sequence number of the record (relative to the data set, including filtering and index ordering); applicable if the table type supports sequence numbers (Paradox only).

iPhyRecNum: type LongInt; specifies the record number for the record; applicable only when the table type supports physical record numbers (dBASE only).

bRecChanged: type Boolean; not currently used.

bSeqNumChanged: type Boolean; not currently used.

bDeleteFlag: type Boolean; indicates whether the record is deleted; applicable only for table types that support soft-deletes (dBASE only).

One of these BDE functions may be invoked in a Delphi application to fill this structure, from which the physical record number may be retrieved. Below is an example of the DbiGetRecord function used for this purpose.

```
function RecNo(ATable: TTable): LongInt;
var
  R: RECProps;
  rslt: DbiResult;
  Error: array [0..255] of Char;
begin
  ATable.UpdateCursorPos;
  rslt := DbiGetRecord(ATable.Handle, dbiNoLock, nil, @R);
  if rslt = DBIERR_NONE then
    Result := R.iPhyRecNum
  else begin
    DbiGetErrorString(rslt, Error);
    ShowMessage(StrPas(Error));
    Result := -1;
  end;
end;
```

As with invoking any BDE function in a Delphi application, the BDE wrapper units DbiTypes, and DbiErrs, DbiProcs must be included in the Uses section of the unit in which the BDE function will be invoked (the Uses section not shown here). To make this function more transportable, it does not reference the subject TTable component directly, but a reference to the TTable is passed as a parameter. If this function is used in a unit that does not reference the Delphi units DB and DBTables, they must be added so that references to the TTable component will be valid.

The UpdateCursorPos method of the TTable is called to ensure that the record number current in the TTable component is synchronized with that of the underlying table.

BDE functions do not in themselves cause an exception if they fail. Rather, they return a value of BDE type DbiResult that indicates the success or failure of the intended operation. This return value must then be retrieved and evaluated by the front-end application, and the appropriate action taken. A result other than DBIERR_NONE indicates an unsuccessful execution of the function. An extra step may be taken (as in the example above) to query the BDE to translate an error code into a readable message with the BDE function DbiGetErrorString. In this example, the return value from the invoking of DbiGetRecord is stored in the variable rslt, and then compared against DBIERR_NONE to determine the success of the function call.

If the call to DbiGetRecord succeeds, the physical record number from the iPhyRecNum field of the pRECProps structure is stored to the variable Result, which is the function's return value. To indicate when the function has failed (i.e., the involing of the DbiGetRecord function failed), a value of negative one is returned instead of the record number. This value is purely arbitrary, and any value of a compatible

type may be used at the discretion of the programmer.

Managing disk volume labels in Delphi

NUMBER : 2854
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Managing disk volume labels in Delphi

This document contains the source code for a unit that is useful for getting, setting, and deleting volume labels from a floppy or hard disk. The code for getting a volume label uses the Delphi FindFirst function, and the code for setting and deleting volume labels involves calling DOS interrupt 21h, functions 16h and 13h respectively. Since function 16h isn't supported by Windows, it must be called through DPMI interrupt 31h, function 300h.

```
{ *** BEGIN CODE FOR VOLLABEL UNIT *** }
unit VolLabel;

interface

uses Classes, SysUtils, WinProcs;

type
  EInterruptError = class(Exception);
  EDPMIError = class(EInterruptError);
  Str11 = String[11];

procedure SetVolumeLabel(NewLabel: Str11; Drive: Char);
function GetVolumeLabel(Drive: Char): Str11;
procedure DeleteVolumeLabel(Drv: Char);

implementation

type
  PRealModeRegs = ^TRealModeRegs;
  TRealModeRegs = record
    case Integer of
      0: (
        EDI, ESI, EBP, EXX, EBX, EDX, ECX, EAX: Longint;
        Flags, ES, DS, FS, GS, IP, CS, SP, SS: Word);
      1: (
        DI, DIH, SI, SIH, BP, BPH, XX, XXH: Word;
        case Integer of
          0: (
            BX, BXH, DX, DXH, CX, CXH, AX, AXH: Word);
          1: (
            BL, BH, BLH, BHH, DL, DH, DLH, DHH,
            CL, CH, CLH, CHH, AL, AH, ALH, AHH: Byte));
  end;

  PExtendedFCB = ^TExtendedFCB;
  TExtendedFCB = Record
```

```
    ExtendedFCBflag : Byte;
    Reserved1       : array[1..5] of Byte;
    Attr            : Byte;
    DriveID         : Byte;
    FileName        : array[1..8] of Char;
    FileExt         : array[1..3] of Char;
    CurrentBlockNum : Word;
    RecordSize      : Word;
    FileSize        : LongInt;
    PackedDate      : Word;
    PackedTime      : Word;
    Reserved2       : array[1..8] of Byte;
    CurrentRecNum   : Byte;
    RandomRecNum    : LongInt;
  end;

procedure RealModeInt(Int: Byte; var Regs: TRealModeRegs);
{ procedure invokes int 31h function 0300h to simulate a real mode }
{ interrupt   from protected mode. }
var
  ErrorFlag: Boolean;
begin
  asm
    mov ErrorFlag, 0        { assume success }
    mov ax, 0300h           { function 300h }
    mov bl, Int             { real mode interrupt to execute }
    mov bh, 0               { required }
    mov cx, 0               { stack words to copy, assume zero }
    les di, Regs            { es:di = Regs }
    int 31h                 { DPMI int 31h }
    jnc @@End                   { carry flag set on error }
  @@Error:
    mov ErrorFlag, 1        { return false on error }
  @@End:
  end;
  if ErrorFlag then
    raise EDPMIError.Create('Failed to execute DPMI interrupt');
end;

function DriveLetterToNumber(DriveLet: Char): Byte;
{ function converts a character drive letter into its numerical equiv. }
begin
  if DriveLet in ['a'..'z'] then
    DriveLet := Chr(Ord(DriveLet) -32);
  if not (DriveLet in ['A'..'Z']) then
    raise EConvertError.CreateFmt('Cannot convert %s to drive number',
                                  [DriveLet]);
  Result := Ord(DriveLet) - 64;
end;

procedure PadVolumeLabel(var Name: Str11);
{ procedure pads Volume Label string with spaces }
var
  i: integer;
begin
  for i := Length(Name) + 1 to 11 do
```

```
    Name := Name + ' ';
end;

function GetVolumeLabel(Drive: Char): Str11;
{ function returns volume label of a disk }
var
  SR: TSearchRec;
  DriveLetter: Char;
  SearchString: String[7];
  P: Byte;
begin
  SearchString := Drive + ':\*.*';
  { find vol label }
  if FindFirst(SearchString, faVolumeID, SR) = 0 then begin
    P := Pos('.', SR.Name);
    if P > 0 then begin                          { if it has a dot... }
      Result := '           ';                   { pad spaces between name }
      Move(SR.Name[1], Result[1], P - 1);      { and extension }
      Move(SR.Name[P + 1], Result[9], 3);
    end
    else begin
      Result := SR.Name;                         { otherwise, pad to end }
      PadVolumeLabel(Result);
    end;
  end
  else
    Result := '';
end;

procedure DeleteVolumeLabel(Drv: Char);
{ procedure deletes volume label from given drive }
var
  CurName: Str11;
  FCB: TExtendedFCB;
  ErrorFlag: WordBool;
begin
  ErrorFlag := False;
  CurName := GetVolumeLabel(Drv);          { get current volume label }
  FillChar(FCB, SizeOf(FCB), 0);           { initialize FCB with zeros }
  with FCB do begin
    ExtendedFCBflag := $FF;                  { always }
    Attr := faVolumeID;                      { Volume ID attribute }
    DriveID := DriveLetterToNumber(Drv); { Drive number }
    Move(CurName[1], FileName, 8);           { must enter volume label }
    Move(CurName[9], FileExt, 3);
  end;
  asm
    push ds                                  { preserve ds }
    mov ax, ss                               { put seg of FCB (ss) in ds }
    mov ds, ax
    lea dx, FCB                              { put offset of FCB in dx }
    mov ax, 1300h                             { function 13h }
    Call DOS3Call                             { invoke int 21h }
    pop ds                                   { restore ds }
    cmp al, 00h                              { check for success }
    je @@End
```

```
  @@Error:                                  { set flag on error }
    mov ErrorFlag, 1
  @@End:
  end;
  if ErrorFlag then
    raise EInterruptError.Create('Failed to delete volume name');
end;

procedure SetVolumeLabel(NewLabel: Str11; Drive: Char);
{ procedure sets volume label of a disk.   Note that this procedure }
{ deletes the current label before setting the new one.   This is }
{ required for the set function to work. }
var
  Regs: TRealModeRegs;
  FCB: PExtendedFCB;
  Buf: Longint;
begin
  PadVolumeLabel(NewLabel);
  if GetVolumeLabel(Drive) <> '' then          { if has label... }
    DeleteVolumeLabel(Drive);                    { delete label }
  Buf := GlobalDOSAlloc(SizeOf(PExtendedFCB));   { allocate real buffer }
  FCB := Ptr(LoWord(Buf), 0);
  FillChar(FCB^, SizeOf(FCB), 0);                { init FCB with zeros }
  with FCB^ do begin
    ExtendedFCBflag := $FF;                      { required }
    Attr := faVolumeID;                          { Volume ID attribute }
    DriveID := DriveLetterToNumber(Drive);       { Drive number }
    Move(NewLabel[1], FileName, 8);              { set new label }
    Move(NewLabel[9], FileExt, 3);
  end;
  FillChar(Regs, SizeOf(Regs), 0);
  with Regs do begin                             { SEGMENT of FCB }
    ds := HiWord(Buf);                           { offset = zero }
    dx := 0;
    ax := $1600;                                 { function 16h }
  end;
  RealModeInt($21, Regs);                        { create file }
  if (Regs.al <> 0) then                         { check for success }
    raise EInterruptError.Create('Failed to create volume label');
end;

end.
{ *** END CODE FOR VOLLABEL UNIT *** }
```

dBASE Expression Indexes: A Primer

NUMBER : 2838
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : dBASE Expression Indexes: A Primer

Indexes for dBASE tables may be based on a the values from a single field, unmodified, or on an expression. Index expressions, unique to dBASE indexes, may be composed of multiple fields, modifications of field values, or combinations of these. The expression for a dBASE expression index is created by using dBASE functions and syntax to concatenate multiple fields or to perform the modifications of field values for fields included in the index expressions.

Two section are included at the end of this technical article which describe the mechanics of creating dBASE expression indexes, one applicable to doing this in the Database Desktop utility and the other for including this capability in Delphi applications.

Expression Indexes Based On Multiple Fields

dBASE functions are available for use in Delphi or the Database Desktop for the express use in index expressions, and then only in conjunction with dBASE indexes. That is, you cannot use dBASE functions or syntax to build an index expression for a Paradox or Local InterBase Server (LIBS) table. Nor can dBASE functions be used in Delphi programming. They are only available for dBASE expression indexes. The dBASE functions and syntax that can be used for expression indexes are provided by the Borland Database Engine (BDE) Dynamic Linked Library (DLL) file IDDBAS01.DLL.

When creating a dBASE index that is to be based on the values from two or more fields in the table for which the index is being created, the two or more fields are concatenated (connected together) in a manner similar to how String type values are concatenated in Delphi syntax: the "+" operator. For example, the expression needed to create an index that orders first on a LastName field and then on a FirstName field would be:

```
LastName + FirstName
```

Unlike in dBASE itself, such indexes that are based on multiple fields are limited to using just those fields in the one table. dBASE allows the creation of indexes based on multiple fields contained in different tables. This is possible only by having the other table open at the time the index is created or when the table containing the index is used.

With multi-field indexes for other table types (e.g., Paradox and InterBase), the multiple fields are delimited by the semi-colon (;), as in:

```
LastName;FirstName
```

In dBASE expression indexes that concatenate multiple fields, an actual expression is used:

```
LastName + FirstName
```

When creating index expressions that concatenate two or more fields, all of the fields included in the index expression must be of the same data type. Additionally, if they are to be concatenated instead of added together, the fields must all be of String type. For example, given two Integer type fields, Value1 and Value2, the index expression...

```
Value1 + Value2
```

...would not cause an error. But then, neither would it concatenate the two field values; it would add them together. Thus, if Value1 for a given record contained 4 and Value2 5, the resulting index node would be an Integer value of 9, not a String concatenation "45".

If fields to be included in an expression index are not of String type, they must be converted. Here are some dBASE functions to convert various data types to String for purposes of creating index expressions:

```
STR(<numeric value> [, <width> [, <decimal places>]])
Converts from either Float or Numeric dBASE types to Character (String)

DTOS(<date value>)
Converts Date value to Character, format YYYYMMDD

MLINE(<memo field>, <line number>)
Extracts a single line from a memo field as a Character value
```

Another consideration in creating indexes based on the concatenation of multiple field is the maximum allowable length of the index value. The value returned by an index expression may not exceed 100 characters. This is a limit on the length of the value returned by the expression, not on the length of the expression itself. For example, you cannot index on the concatenation of two fields that both have a length of 255 characters.

Expression Indexes Based On Modifications Of Field Values

In addition to creating indexes based on the concatenation of two or more field values, it is also possible to construct an index that is based on a modification of a field value. Examples of this include indexing on just the first three characters of a String type field, on just the year and month from a Date field, indexing on a contantenation of a String and Date field such that the ordering of the String field is ascending but the Date descending, and even indexing on Boolean field values.

Creating indexes that are based on modifications of field values requires at least a working knowledge of dBASE functions and syntax -- because the process uses dBASE, and not Delphi, functions and syntax. The dBASE function SUBSTR() extracts a substring of a String value. The Delphi equivalent for this dBASE function is Copy. But, of these two functions that serve the same purpose, only SUBSTR() may be used in dBASE index expressions.

To use dBASE functions in dBASE index expressions, simply include the function wherever an index expression is called for, using the appropriate dBASE syntax for the function, along with a reference to the name(s) of the field(s) used in the function. For example, an index expression based on only the last three characters of a String type field called Code, that is 20 characters long, would be:

```
RIGHT(Code, 3)
```

Caution should be used in constructing dBASE index expressions that modify field values to ensure that the resulting expression would return a value of a consistent length for every record in the table. For instance, the dBASE TRIM() function removes the trailing blanks (ASCII decimal 32) from a String type value. If this were used in conjunction with concatenating two String fields where the field does not contain values of the same length for all records, the value resulting from the expression will not be the same for all records. Case in point, an index expression based on the concatenation of a LastName and a FirstName field, where the TRIM() function is applied to the LastName field:

```
TRIM(LastName) + FirstName
```

This expression would not return values of a consistent length for all records. If the LastName and FirstName fields contained the values...

```
LastName FirstName
-------- ---------
Smith    Jonas
Wesson   Nancy
```

...the result of applying the index expression above would be:

```
SmithJonas
WessonNancy
```

As can be seen, the length of the value for the first record would be 10 characters, while that for the second 11 characters. The index nodes created for this index expression would be based on the field values for the first record encountered. This would result in an index node 10 characters long being applied to the field values for all record. In this example, that would result in the truncation of the expression value for the second record to "WessonNanc". This would subsequently cause searches based on the full field value to fail.

The solution to this dilemma would be to not use the TRIM() function so that the full length of the LastName field, including padding from the trailing spaces, is used. In indexes that use the IIF() function to order by one field or another, based on the evaluation of a logical expression in the IIF(), if the two fields are of different lengths, the shorter field would need to be padded with spaces to make it the same length as the longer field. For example, assuming an index that uses the IIF() function to index either on a Company or a Name field, based on the contents of Category field, and where the Company field is 40 characters long but the Name field is 25 characters long, the Name field would need to be padded with 15 spaces; accomplished with the dBASE function SPACE(). That

index expression would then be:

```
IIF(Category = "B", Company, Name + SPACE(15))
```

Searches And dBASE Expression Indexes

dBASE expression indexes are exceptions to the norm in how they are handled by Delphi and the BDE as opposed to how multiple field indexes for other table types are handled.

This puts such dBASE indexes into a separate class. Handling of such indexes by Delphi and the BDE is different than those for other table types. One of these differences is that not all index-based searching using Delphi syntax can be used with these dBASE expression indexes. The FindKey, FindNearest, and GotoKey methods of the TTable component cannot be used with expression indexes. If an attempt to use FindKey is made, this will result in the error message: "Field index out of range." If the GotoKey method is tried, this error message may occur or the record pointer may just not move (indicating the search value was not found). Only the GotoNearest method may be used with expression indexes. Even then, the GotoNearest method may not work with some index expressions. Experimentation will be needed to see whether the GotoNearest method will work with a given index expression.

Filtering With dBASE Expression Indexes

As with index-based searches, dBASE expression indexes present some exceptions when using Delphi filtering.

With an expression index active, the SetRange method of the TTable component will produce the error: "Field index out of range." However, with the same expression index active, the SetRangeStart and SetRangeEnd methods will successfully filter the data set.

For example, with an expression index concatenating a LastName and a FirstName field active, the code below using the FindKey method (intended to filter to just those records where the first character of the LastName field is "S") will fail with an error:

```
begin
  Table1.SetRange(['S'], ['Szzz'])
end;
```

Whereas, the code below, with the same expression index active and filtering on the same LastName field, will successfully filter the data and not incur an error:

```
begin
  with Table1 do begin
    SetRangeStart;
    FieldByName('LastName').AsString := 'S';
    SetRangeEnd;
    FieldByName('LastName').AsString := 'Szzz';
    ApplyRange;
```

```
    end;
  end;
```

And, as is the case with index-based searches, with filtering, success of a filtering attempt will also be dependent on the index expression. The use of the SetRangeStart and SetRangeEnd methods in the preceeding example worked with an index that simply concatenated two String type fields. But if the expression for the index was instead based conditionally on one or the other fields through use of the IIF() function, the same filtering routine would fail (although without an error).

Some Handy dBASE Index Expressions

Here are some handy dBASE index expressions. Some are readily apparent in the intended purpose, others are more arcane.

Character field ascending and Date field descending

With a Character field called Name and a Date field OrdDate:

```
Name + STR(OrdDate - {12/31/3099}, 10, 0)
```

Character field ascending and Numeric (or Float) field descending

With a Character field called Company and a Numeric field Amount (the Amount field being 9 digits wide with two decimal places):

```
Company + STR(Amount - 999999.99, 9, 2)
```

Ordering by a Logical field

To have True values appear before False values for a Logical field called Paid:

```
IIF(Paid, "A", "Z")
```

Two Numeric (or Float) fields

Assuming two Numeric fields with widths of five and two decimal places, the first field named Price and the second Quantity:

```
STR(Price, 5, 2) + STR(Quantity, 5, 2)
```

Ordering by one field of two, depending on a logical condition

Ordering by the names of months in a Character field

Assuming a field containing the names of the months ("Jan," "Feb" etc.) to put the records in proper month order (field named M):

```
IIF(M="Jan", 1, IIF(M="Feb", 2, IIF(M="Mar", 3, IIF(M="Apr", 4,
IIF(M="May", 5, IIF(M="Jun", 6, IIF(M="Jul", 7, IIF(M="Aug", 8,
IIF(M="Sep", 9, IIF(M="Oct", 10, IIF(M="Nov", 11, 12)))))))))))
```

(The above is a single expression line, broken into multiple lines here due to page width.)

Ordering by the first line of a memo field

For a memo field named Notes:

```
MLINE(Notes, 1)
```

Ordering by the middle three characters in a nine character long field

For a nine character long field called StockNo:

```
SUBSTR(StockNo, 4, 3)
```

Creating dBASE Expression Indexes In Database Desktop

In the Database Desktop utility, indexes may be created for a table either duting the process of creating a new table or by restructuring an existing table. In both cases, the Define Index dialog is used to create one or more indexes for the table used.

To get to the Create Index dialog while creating a new table, in the Create dBASE Table dialog (showing the structure), from the Table Properties listbox select "Indexes" and click the Define button.

To get to the Create Index dialog to create an index for an existing table, select Utilities|Restructure, select the table file from the Select File dialog, and in the Restructure dBASE Table dialog (showing the table structure) from the Table Properties listbox select "Indexes" and click the Define button.

Once in the Create Index dialog, expression indexes can be created by clicking the Expression Index button and entering the expression to be used in the Expression Index entry field. To assist in this process, you can double-click on a field name in the Field List listbox and that field name will be inserted into the Index Expression entry field at the current insertion point (caret position).

Once the index expression has been entered, click the OK button. Enter the name of the new index tag in the Index Tag Name entry field on the Save Index As dialog and click OK. (Remember, dBASE index tag names cannot exceed ten characters in length and must abide by the normal dBASE naming conventions.)

Creating dBASE Expression Indexes In Delphi Applications

dBASE indexes can be created programmatically in Delphi applications, either as a new table is being created (CreateTable method of the TTable component) or by adding an index to an existing table.

Creating an index as part of a new table being created is a matter of calling the Add method for the IndexDefs property of the TTable. A special consideration that the index options must include the option ixExpression. This index option is unique to dBASE indexes, and should only be used with dBASE expression indexes. For example:

```
with Table1 do begin
  Active := False;
  DatabaseName := 'Delphi_Demos';
  TableName := 'CustInfo';
  TableType := ttdBASE;
  with FieldDefs do begin
    Clear;
    Add('LastName', ftString, 30, False);
    Add('FirstName', ftString, 20, False);
  end;
  with IndexDefs do   begin
    Clear;
    Add('FullName', 'LastName + FirstName', [ixExpression]);
  end;
  CreateTable;
end;
```

Adding an index to an existing table is accomplished by calling the Add-Index method of the TTable. Again, the index options must include the TIndexOptions value ixExpression.

```
Table1.AddIndex('FullName', 'LastName + FirstName', [ixExpression]);
```

Learning More About dBASE Functions And Syntax

Only dBASE functions and syntax that apply to data manipulation can be used to construct a dBASE expression index. However, it is beyond the scope of this technical article to fully list and describe all of these functions. To learn more about dBASE data manipulation functions, the user is advised to consult the dBASE Language Reference manual or one of the many third-party dBASE books.

How to terminate all running applications

NUMBER : 2856
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to terminate all running applications

Q: How do I terminate all running tasks?

A: Below is some code that will help if you want to terminate ALL tasks, no questions asked.

A word of caution, before you run this for the first time, make sure that you save it and anything else that may have some pending data.

```
procedure TForm1.ButtonKillAllClick(Sender: TObject);
var
  pTask     : PTaskEntry;
  Task      : Bool;
  ThisTask: THANDLE;
begin
  GetMem (pTask, SizeOf (TTaskEntry));
  pTask^.dwSize := SizeOf (TTaskEntry);

  Task := TaskFirst (pTask);
  while Task do
  begin
    if pTask^.hInst = hInstance then
      ThisTask := pTask^.hTask
    else
      TerminateApp (pTask^.hTask, NO_UAE_BOX);
    Task := TaskNext (pTask);
  end;
  TerminateApp (ThisTask, NO_UAE_BOX);
end;
```

How to check to see if a drive is ready.

NUMBER : 2857
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to check to see if a drive is ready.

Q: How can I check to see if there is a disk in the "A" drive without an error message box telling you that it is not ready?

A: The following function accepts a drive letter as a parameter, and it will return a boolean value that indicates whether or not there is a disk in the drive.

```
function DiskInDrive(Drive: Char): Boolean;
var
  ErrorMode: word;
begin
  { make it upper case }
  if Drive in ['a'..'z'] then Dec(Drive, $20);
  { make sure it's a letter }
  if not (Drive in ['A'..'Z']) then
    raise EConvertError.Create('Not a valid drive ID');
  { turn off critical errors }
  ErrorMode := SetErrorMode(SEM_FailCriticalErrors);
  try
    { drive 1 = a, 2 = b, 3 = c, etc. }
    if DiskSize(Ord(Drive) - $40) = -1 then
      Result := False
    else
      Result := True;
  finally
    { restore old error mode }
    SetErrorMode(ErrorMode);
  end;
end;
```

How to do pointer arithmetic in Delphi.

NUMBER : 2858
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to do pointer arithmetic in Delphi.

Q: How do I do pointer arithmetic in Delphi?

A: First a brief explanation of pointer arithmetic. When you are dealing with dynamic memory locations and all you have is a pointer to where it all begins, you want to have the ability to traverse that line of memory to be able to perform whatever functions you have in mind for that data. This can be accomplished by changing the place in memory where the pointer points. This is called pointer arithmetic.

The main idea that must be kept in mind when doing your pointer arithmetic is that you must increment the pointer's value by the correct amount. (The correct amount is determined by the size of the object receiving the pointer. e.g. char = 1 byte; integer = 2 bytes; double = 8 bytes; etc.) The Inc() and Dec() functions will alter the amount by the correct amount. (The compiler knows what the correct size is.)

For an example of the practical application of pointer arithmetic, download the BreakApart() TI2905.

If you are doing dynamic memory allocation, it is done like this:

```
uses WinCRT;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyArray: array[0..30] of char;
  b: ^char;
  i: integer;
begin
  StrCopy(MyArray, 'Lloyd is the greatest!'); {get something to point to}
  b := @MyArray; { assign the pointer to the memory location }
  for i := StrLen(MyArray) downto 0 do
  begin
    write(b^);    { write out the char at the current pointer location. }
    inc(b);       { point to the next byte in memory }
  end;
end;
```

The following code demonstrates that the Inc() and Dec() functions will increment or decrement accordingly by size of the type the pointer points to:

```
var
```

```
  P1, P2 : ^LongInt;
  L : LongInt;
begin
  P1 := @L; { assign both pointers to the same place }
  P2 := @L;
  Inc(P2);   { Increment one }

{ Here we get the difference between the offset values of the
two pointers.   Since we originally pointed to the same place in
memory, the result will tell us how much of a change occured
when we called Inc(). }

  L := Ofs(P2^) - Ofs(P1^); { L = 4; i.e. sizeof(longInt) }
end;
```

You can change the type to which P1 and P2 point to something other than a longint to see that the change is always the correct value (SizeOf(P1^)).

How to do bit-wise manipulation.

NUMBER : 2859
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to do bit-wise manipulation.

Q: How do I do bit-wise manipulation?

A:

```
{*****************************************
TheBit parameter is counted from 0..31
*****************************************}

unit Bitwise;

interface
   function IsBitSet(const val: longint; const TheBit: byte): boolean;
   function BitOn(const val: longint; const TheBit: byte): LongInt;
   function BitOff(const val: longint; const TheBit: byte): LongInt;
   function BitToggle(const val: longint; const TheBit: byte): LongInt;



implementation

function IsBitSet(const val: longint; const TheBit: byte): boolean;
begin
   result := (val and (1 shl TheBit)) <> 0;
end;

function BitOn(const val: longint; const TheBit: byte): LongInt;
begin
   result := val or (1 shl TheBit);
end;

function BitOff(const val: longint; const TheBit: byte): LongInt;
begin
   result := val and ((1 shl TheBit) xor $FFFFFFFF);
end;

function BitToggle(const val: longint; const TheBit: byte): LongInt;
begin

   result := val xor (1 shl TheBit);
end;

end.
```

Basic Delphi DLL template

NUMBER : 2860
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : May 21, 1996

TITLE : Basic Delphi DLL template

DLL sample

Without units

First the DLL "framework" that you wanted, save as DLLFRAME.DPR:

```
{--------------------DLLFRAME.DPR--------------------------}
library Dllframe;

uses WinTypes;

function   GetString : string ; export ;
begin
  Result := 'Hello from the DLL!' ;
end;

exports
  GetString;

begin
end.
{----------------------------------------------------------}
```

Now here's the calling program, save it as DLLCALL.DPR:

```
{--------------------DLLCALL.DPR---------------------------}
program Dllcall;

uses
  Dialogs;

{$R *.RES}

function GetString : string ; far ; external 'DLLFRAME' ;

begin
  MessageDlg( GetString, mtInformation, [ mbOK ], 0 ) ;
end.
```

With units

Here's the calling program, save it as DLLCALL.DPR:

```
{--------------------DLLCALL.DPR---------------------------}
```

```
program Dllcall;

uses
  Dialogs;

{$R *.RES}

function GetString : string ; far ; external 'MyDLL' ;


begin
  MessageDlg( GetString, mtInformation, [ mbOK ], 0 ) ;
end.
{-------------------------------------------------------------}
```

The DLL "framework" that you wanted, save as DLLFRAME.DPR:

```
{--------------------DLLFRAME.DPR-------------------------}
library Dllframe;

uses DLLUnit;

exports
  GetString;

begin
end.
{-------------------------------------------------------------}
```

The unit we will save as dllunit.pas:

```
{--------------------dllunit.pas-------------------------}

unit DLLUnit;
interface

uses WinTypes;

function GetString: string; export;

implementation

function GetString: string;
begin
  GetString := 'Hello from the DLL!' ;
end ;

begin
end.
```

Form display with different screen resolutions.

NUMBER : 2861
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Form display with different screen resolutions.

When designing forms, it is sometimes helpful to write the code so that the screen and all of its objects are displayed at the same size no matter what the screen resolution is. Here is some code to show how that is done:

```
implementation
const
  ScreenWidth: LongInt = 800; {I designed my form in 800x600 mode.}
  ScreenHeight: LongInt = 600;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  scaled := true;
  if (screen.width <> ScreenWidth) then
  begin
    height := longint(height) * longint(screen.height) div ScreenHeight;
    width := longint(width) * longint(screen.width) div ScreenWidth;
    scaleBy(screen.width, ScreenWidth);
  end;
end;
```

Then, you will want to have something that checks to see that the font sizes are OK. You can iterate over each child control's font to adjust its size as necessary. This can be done as follows:

```
type
  TFooClass = class(TControl); { needed to get at protected }
                               { font property }

var
  i: integer;
begin
  for i := ControlCount - 1 downto 0 do
    TFooClass(Controls[i]).Font.Size :=
        (NewFormWidth div OldFormWidth) *
        TFooClass(Controls[i]).Font.Size;
end;
```

Note: The following are issue to bear in mind when scaling Delphi applications (forms) on different screen resolutions:

* Decide early on in the form design stage whether you're going to allow the form to be scaled or not. The advantage of not scaling is that nothing changes at runtime. The disadvantage of not scaling is that nothing changes at runtime (your form may be far too small or too large to read on some systems if it is not scaled).

* If you're NOT going to scale the form, set Scaled to False.

* Otherwise, set the Form's Scaled property to True.

* Set AutoScroll to False. AutoScroll = True means 'don't change the form's frame size at runtime' which doesn't look good when the form's contents do change size.

* Set the form's font to a scaleable TrueType font, like Arial. MS San Serif is an ok alternate, but remember that it is still a bitmapped font. Only Arial will give you a font within a pixel of the desired height. NOTE: If the font used in an application is not installed on the target computer, then Windows will select an alternative font within the same font family to use instead. This font may not match the same size of the original font any may cause problems.

* Set the form's Position property to something other than poDesigned. poDesigned leaves the form where you left it at design time, which for me always winds up way off to the left on my 1280x1024 screen - and completely off the 640x480 screen.

* Don't crowd controls on the form - leave at least 4 pixels between controls, so that a one pixel change in border locations (due to scaling) won't show up as ugly overlapping controls.

* For single line labels that are alLeft or alRight aligned, set AutoSize to True. Otherwise, set AutoSize to False.

* Make sure there is enough blank space in a label component to allow for font width changes - a blank space that is 25% of the length of the current string display length is a little too much, but safe. (You'll need at least 30% expansion space for string labels if you plan to translate your app into other languages) If AutoSize is False, make sure you actually set the label width appropriately. If AutoSize is True, make sure there is enough room for the label to grow on its own.

* In multi-line, word-wrapped labels, leave at least one line of blank space at the bottom. You'll need this to catch the overflow when the text wraps differently when the font width changes with scaling. Don't assume that because you're using large fonts, you don't have to allow for text overflow - somebody else's large fonts may be larger than yours!

* Be careful about opening a project in the IDE at different resolutions. The form's PixelsPerInch property will be

modified as soon as the form is opened, and will be saved to the DFM if you save the project. It's best to test the app by running it standalone, and edit the form at only one resolution. Editing at varying resolutions and font sizes invites component drift and sizing problems.

* Speaking of component drift, don't rescale a form multiple times, at design time or a runtime. Each rescaling introduces roundoff errors which accumulate very quickly since coordinates are strictly integral. As fractional amounts are truncated off control's origins and sizes with each successive rescaling, the controls will appear to creep northwest and get smaller. If you want to allow your users to rescale the form any number of times, start with a freshly loaded/created form before each scaling, so that scaling errors do not accumulate.

* Don't change the PixelsPerInch property of the form, period.

* In general, it is not necessary to design forms at any particular resolution, but it is crucial that you review their appearance at 640x480 with small fonts and large, and at a high-resolution with small fonts and large before releasing your app. This should be part of your regular system compatibility testing checklist.

* Pay close attention to any components that are essentially single-line TMemos - things like TDBLookupCombo. The Windows multi-line edit control always shows only whole lines of text - if the control is too short for its font, a TMemo will show nothing at all (a TEdit will show clipped text). For such components, it's better to make them a few pixels too large than to be one pixel too small and show not text at all.

* Keep in mind that all scaling is proportional to the difference in the font height between runtime and design time, NOT the pixel resolution or screen size. Remember also that the origins of your controls will be changed when the form is scaled - you can't very well make components bigger without also moving them over a bit.

How to keep the app iconized.

NUMBER : 2862
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to keep the app iconized.

iconized apps

Q: How do I keep the form in icon form when I run it?

A:

1. You must set WindowState to wsMinimized in the form's properties.

2. In the private section of the form object's declaration, put:

```
PROCEDURE WMQueryOpen(VAR Msg : TWMQueryOpen); message WM_QUERYOPEN;
```

3. In the implementation section, put this method:

```
PROCEDURE TForm1.WMQueryOpen(VAR Msg : TWMQueryOpen);
begin
  Msg.Result := 0;
end;
```

That's it! The form will always remain iconic.

How to use a custom cursor.

NUMBER : 2863
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to use a custom cursor.

Q: How do I use one of the cursor files in the c:\delphi\images\cursors?

A: Use the image editor to load the cursor into a RES file.
The following example assumes that you saved the cursor in the RES file as "cursor_1", and you save the RES file as MYFILE.RES.

```
(*** BEGIN CODE ***)
{$R c:\programs\delphi\MyFile.res}     { This is your RES file }

const PutTheCursorHere_Dude = 1;        { arbitrary positive number }

procedure stuff;
begin
   screen.cursors[PutTheCursorHere_Dude] := LoadCursor(hInstance,
                                                       PChar('cursor_1'));
   screen.cursor := PutTheCursorHere_Dude;
end;
```

How to use a form several times

NUMBER : 2896
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : How to use a form several times

multiple forms

Q: I have a form that is a sort of template. I want to be able to create and show the same form several times (with different data in the fields). How do I use the same form several times?

A: You need to make modeless window by calling create and show for each form instance, like this:

```
with TMyForm.create(self) do show;
```

To demonstrate how to use and control these new forms, here is an example that changes the caption and name of each form that is created. You have access to it through the form's component array. This example uses an about box (named "box") as the other form. Also, there is a variable called "TheFormCount" that keeps track of how many times the form is instantiated.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with TBox.create(self) do begin
    Name := 'AboutBox_' + intToStr(TheFormCount);
    caption := 'About Box # ' + intToStr(TheFormCount);
    Show;
  end;
  inc(TheFormCount);
end;
```

These forms can be found and used by their name by means of the FindComponent method used something like this:

```
with Form1.FindComponent('AboutBox_' + IntToStr(Something)) as TForm do
    DoSomethingHere;
```

How to set a max and min form size.

NUMBER : 2865
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to set a max and min form size.

When you want to control how much your users can resize your form, you can control that by setting the MinMax values. (If you use the resize method to limit the size, it will work, but it won't look quite as good.)

Note: To make it so that the user cannot change the form's size at all, make the min and max sizes the same values.

This is an example of how to declare and use the wm_GetMinMaxInfo windows message in your applications.

```
unit MinMax;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
    procedure WMGetMinMaxInfo(var MSG: Tmessage); message WM_GetMinMaxInfo;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.WMGetMinMaxInfo(var MSG: Tmessage);
Begin
  inherited;
  with PMinMaxInfo(MSG.lparam)^ do
  begin
    with ptMinTrackSize do
    begin
      X := 300;
      Y := 150;
    end;
```

```
      with ptMaxTrackSize do
      begin
        X := 350;
        Y := 250;
      end;
    end;
end;

end.
```

How to get the windows and DOS versions.

NUMBER : 2866
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to get the windows and DOS versions.

Q: How can I get the Windows or DOS version numbers?

A: The API call GetVersion will do it, but the information is encrypted into a longint. Here is how to get and decrypt the information:

```
Type
  TGetVer = record
    WinVer,
    WinRev,
    DosRev,
    DosVer: Byte;
  end;

const
  VerStr = '%d.%d';

procedure TForm1.Button1Click(Sender: TObject);
var
  AllVersions: TGetVer;
begin
  AllVersions := TGetVer(GetVersion);
  Edit1.Text := Format(VerStr, [AllVersions.WinVer, AllVersions.WinRev]);
  Edit2.Text := Format(VerStr, [AllVersions.DOSVer, AllVersions.DOSRev]);
end;
```

Note1: The values that windows displays for the versions and the values that it returns through its API call are not always the same. e.g. The workgroup version displays as 3.10 rather than 3.11.

Note2: Win32 applications should call GetVersionEx rather than GetVersion.

How to tell what kind of drive is used.

NUMBER : 2867
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to tell what kind of drive is used.

When dealing with multiple drives, it is helpful to know whether a drive is associated with a is attached to a letter (A, B, C, etc), and what its type is. This code uses the API GetDriveType function to do that.

```
function ShowDriveType(DriveLetter: char): string;
var
  i: word;
begin
  if DriveLetter in ['A'..'Z'] then {Make it lower case.}
    DriveLetter := chr(ord(DriveLetter) + $20);
  i := GetDriveType(ord(DriveLetter) - ord('a'));
  case i of
    DRIVE_REMOVABLE: result := 'floppy';
    DRIVE_FIXED: result := 'hard disk';
    DRIVE_REMOTE: result := 'network drive';
    else result := 'does not exist';
  end;
end;
```

How to determine the current record number.

NUMBER : 2869
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to determine the current record number.

Q: "How can I determine the current record number for a dataset?"

A: If the dataset is based upon a Paradox or dBASE table then the record number can be determined with a couple of calls to the BDE (as shown below). The BDE doesn't support record numbering for datasets based upon SQL tables, so if your server supports record numbering you will need to refer to its documentation.

The following function is given as part of a whole unit and takes as its parameter any component derived from TDataset (i.e. TTable, TQuery, TStoredProc) and returns the current record number (greater than zero) if it is a Paradox or dBASE table. Otherwise, the function returns zero.

NOTE: for dBASE tables the record number returned is always the physical record number. So, if your dataset is a TQuery or you have a range set on your dataset then the number returned won't necessarily be relative to the dataset being viewed, rather it will be based on the record's physical position in the underlying dBASE table.

```
uses
  DB, DBTables, DbiProcs, DbiTypes, DbiErrs;

function GetRecordNumber(Dataset: TDataset): Longint;
var
  CursorProps: CurProps;
  RecordProps: RECProps;
begin
  { Return 0 if dataset is not Paradox or dBASE }
  Result := 0;
  with Dataset do
  begin
    { Is the dataset active? }
    if State = dsInactive then
      raise EDatabaseError.Create('Cannot perform this operation '+
                                  'on a closed dataset');

    { We need to make this call to grab the cursor's iSeqNums }
    Check(DbiGetCursorProps(Handle, CursorProps));

    { Synchronize the BDE cursor with the Dataset's cursor }
```

```
      UpdateCursorPos;

      { Fill RecordProps with the current record's properties }
      Check(DbiGetRecord(Handle, dbiNOLOCK, nil, @RecordProps));

      { What kind of dataset are we looking at? }
      case CursorProps.iSeqNums of
        0: Result := RecordProps.iPhyRecNum;   { dBASE     }
        1: Result := RecordProps.iSeqNum;        { Paradox }
      end;
    end;
end;

end.
```

How to automate logon for Paradox tables

NUMBER : 2870
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 1, 1996

TITLE : How to automate logon for Paradox tables

Password automation

Q: I have a paradox table that uses a password. How do I make it so that the form that uses the table comes up without prompting the user for the password?

A: The table component's ACTIVE property must be set to FALSE (If it is active before you have added the pasword, you will be prompted). Then, put this code in the handler for the form's OnCreate event:

```
Session.AddPassword('My secret password');
Table1.Active := True;
```

Once you close the table, you can remove the password with RemovePassword('My secret password'), or you can remove all current passwords with RemoveAllPasswords. (Note: This is for Paradox tables only.)

packing a dBASE table

NUMBER : 2873
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : May 21, 1996

TITLE : packing a dBASE table

packing a dBASE table

Q: How do I pack a dBASE table?

A: To pack a dBASE table that has been opened with a TTable, use the BDE function DbiPackTable. There are two basic steps to do this:

1. Add the following units to your uses clause:

{ For Delphi 1.0: } DBITYPES, DBIPROCS and DBIERRS;
{ For Delphi 2.0: } BDE;

2) Then call the DbiPackTable BDE function as follows:

```
Check(DbiPackTable(Table1.DbHandle, Table1.Handle, Nil, szDBASE, TRUE));
```

Notes:
* The table must be opened in exclusive mode.
* Use the Check procedure when calling BDE API functions. Check will raise an exception if an error occurs on the BDE call.

string manipulation routines

NUMBER : 2892
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : May 21, 1996

TITLE : string manipulation routines

Here are some standard string manipulation functions:

```
{To determine if the character is a digit.}
function IsDigit(ch: char): boolean;
begin
  Result := ch in ['0'..'9'];
end;

{To determine if the character is an uppercase letter.}
function IsUpper(ch: char): boolean;
begin
  Result := ch in ['A'..'Z'];
end;

{To determine if the character is an lowercase letter.}
function IsLower(ch: char): boolean;
begin
  Result := ch in ['a'..'z'];
end;

{Changes a character to an uppercase letter.}
function ToUpper(ch: char): char;
begin
  Result := chr(ord(ch) and $DF);
end;

{Changes a character to a lowercase letter.}
function ToLower(ch: char): char;
begin
  Result := chr(ord(ch) or $20);
end;

{ Capitalizes first letter of every word in s }
function Proper(const s: string): string;
var
  i: Integer;
  CapitalizeNextLetter: Boolean;
begin
  Result := LowerCase(s);
  CapitalizeNextLetter := True;
  for i := 1 to Length(Result) do
  begin
    if CapitalizeNextLetter and IsLower(Result[i]) then
      Result[i] := ToUpper(Result[i]);
    CapitalizeNextLetter := Result[i] = ' ';
```

```
  end;
end;

{ NOTE: The following functions are available in Delphi 2.0,
  but not in Delphi 1.0. }

{Supresses trailing blanks in a string.}
function TrimRight(const s: string): string;
var
  i: integer;
begin
  i := Length(s);
  while (I > 0) and (s[i] <= ' ') do Dec(i);
  Result := Copy(s, 1, i);
end;

{Removes the leading spaces from a string.}
function TrimLeft(const S: string): string;
var
  I, L: Integer;
begin
  L := Length(S);
  I := 1;
  while (I <= L) and (S[I] <= ' ') do Inc(I);
  Result := Copy(S, I, Maxint);
end;

{ Removes leading and trailing whitespace from s);
function Trim(const S: string): string;
var
  I, L: Integer;
begin
  L := Length(S);
  I := 1;
  while (I <= L) and (S[I] <= ' ') do Inc(I);
  if I > L then Result := '' else
  begin
    while S[L] <= ' ' do Dec(L);
    Result := Copy(S, I, L - I + 1);
  end;
end;
```

WinExecAndWait

NUMBER : 2894
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : WinExecAndWait

Q: How do I execute a program and have my code wait until it is finished?

A:

```
uses Wintypes,WinProcs,Toolhelp,Classes,Forms;

Function WinExecAndWait(Path : string; Visibility : word) : word;
var
  InstanceID : THandle;
  PathLen : integer;
begin
  { inplace conversion of a String to a PChar }
  PathLen := Length(Path);
  Move(Path[1],Path[0],PathLen);
  Path[PathLen] := #0;
  { Try to run the application }
  InstanceID := WinExec(@Path,Visibility);
  if InstanceID < 32 then { a value less than 32 indicates an Exec error }
     WinExecAndWait := InstanceID
  else
  begin
    Repeat
      Application.ProcessMessages;
    until Application.Terminated or (GetModuleUsage(InstanceID) = 0);
    WinExecAndWait := 32;
  end;
end;
```

How to check for app already running.

NUMBER : 2895
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : May 21, 1996

TITLE : How to check for app already running.

Q: How can I write my Delphi program to detect if there is already another copy running and exit if so?

A: Create a unit called PrevInst and add it to your uses clause. Here's the code:

```
unit PrevInst;

interface

uses
  WinTypes, WinProcs, SysUtils;

type
  PHWND = ^HWND;
  function EnumFunc(Wnd:HWND; TargetWindow:PHWND): bool; export;
  procedure GotoPreviousInstance;

implementation

  function EnumFunc(Wnd:HWND; TargetWindow:PHWND): bool;
  var
    ClassName : array[0..30] of char;
  begin
    Result := true;
    if GetWindowWord(Wnd,GWW_HINSTANCE) = hPrevInst then
    begin
      GetClassName(Wnd,ClassName,30);
      if StrIComp(ClassName,'TApplication') = 0 then
      begin
        TargetWindow^ := Wnd;
        Result := false;
      end;
    end;
  end;

  procedure GotoPreviousInstance;
  var
    PrevInstWnd : HWND;
  begin
    PrevInstWnd := 0;
    EnumWindows(@EnumFunc,longint(@PrevInstWnd));
    if PrevInstWnd <> 0 then
      if IsIconic(PrevInstWnd) then
        ShowWindow(PrevInstWnd, SW_RESTORE)
```

```
        else
          BringWindowToTop(PrevInstWnd);
  end;

  end.
```

And then make the main block of your *.DPR file look
something like this--

```
begin
  if hPrevInst <> 0 then
    GotoPreviousInstance
  else
  begin
    Application.CreateForm(MyForm, MyForm);
    Application.Run;
  end;
end.
```

How to use a popup menu with a VBX.

NUMBER : 2864
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : How to use a popup menu with a VBX.

Q: I want to be able to right click on my VBX and have a popup menu display. When I use a popup menu for the form, it shows no matter where I right click. I want to just have it popup for right clicks on the vbx.

How do I trap for that?

A: Here it is:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if button = mbRight then
    with (Sender AS TControl) do
      with ClientToScreen(Point(X,Y)) do
      begin
        PopupMenu1.PopupComponent := TComponent(Sender);
        PopupMenu1.Popup(X,Y);
      end;
end;
```

Note: The form's PopupMenu property must be empty, or it will popup from everywhere. If you want the form to be the only place showing the popup, place this method on the form's OnMouseDown event. If you want the VBX to be the only place, then place it on the VBX's OnMouseDown event, etc.

Manually Installing Delphi

NUMBER : 2899
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : February 28, 1996

TITLE : Manually Installing Delphi

To manually install Delphi you should first copy the entire run-image directory off of the CD-ROM onto you hard-drive. This is one of the main directories on the delphi CD-ROM. This includes copying all the files from the run-image\windows to your local \windows directory and the \windows\system files into your \windows\system directory, UNLESS you have NT. If you have NT you will want to do as above, but do not overwrite any existing files.

It is important that if you have another Borland DataBase product that you Z-copy the \idapi directory from the run-image ONTO the existing \idapi directory on your computer.

To manually install delphi you first need to add the following lines to your win.ini
(this is all dependent on which drive/directory that you copied the run-image to):

```
[IDAPI]
DLLPATH=C:\IDAPI
CONFIGFILE01=C:\IDAPI\IDAPI.CFG

[Borland Language Drivers]
LDPath=C:\IDAPI\LANGDRV


[BWCC]
BitmapLibrary=BWCC.DLL

[Interbase]
RootDirectory=C:\IBLOCAL

[Paradox Engine]
UserName=PxEngine
NetNamePath=C:\
RecBufs=64
MaxLocks=64
MaxFiles=64
SwapSize=64

[DDE Servers]
DBD=C:\DBD\DBD

[DBD]
WORKDIR=C:\DBD
PRIVDIR=C:\DBD\DBDPRIV
```

You will also want to add the following lines to your
autoexec.bat file:

```
rem add the following if it they are not already present
rem you only need share if using windows 3.1

Share /f:4096 /l:40
path=%path%;c:\iblocal\bin;
```

These are once again dependent on the exact drive/directory
that you
copied the run-image to.

You will now want to download the file dlpgrp.zip from
our download BBS or from the web site.
This file will create the Icons and the Delphi Program Group.

Printing in the DOS IDE under Windows 95

NUMBER : 2901
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : Printing in the DOS IDE under Windows 95

Printing from the DOS IDE in Windows 95

To print from the Borland Pascal IDE under Windows 95 can require more than just selecting print from the file menu. Follow this step by step procedure if you can print from Windows applications but cannot print from the DOS IDE.

1.Open the printers folder under the control panel sub-directory. You can get to the control panel by going from the start button to the settings then click control panel. Inside control panel double click the printers folder.

2.Click the icon for the printer you are going to print on.

3.On the file menu click properties.

4.Click on the details tab and then click on the Capture Printer Port button.

5.In the Device list, select the printer port that you want to capture.

6.Type the network path to the printer, and then click OK to save the selection.

7.In the Print To The Following Port dialog box, select the port you just mapped.

For a handy additional reference on the topic also see the Windows help file under "Capture".

Different colored characters in a string grid

NUMBER : 2903
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : Different colored characters in a string grid

This unit will show how to have text in a string grid where the
characters are different colors.

```
unit Strgr;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Grids, StdCtrls, DB;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    procedure StringGrid1DrawCell(Sender: TObject; Col, Row: Longint;
      Rect: TRect; State: TGridDrawState);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col, Row: Longint;
  Rect: TRect; State: TGridDrawState);
const
  CharOffset = 3;
begin
  with StringGrid1.canvas do
  begin
    font.color := clMaroon;
    textout(rect.left + CharOffset, rect.top + CharOffset, 'L');
      font.color := clNavy;
    textout(rect.left + CharOffset + TextWidth('L'),
      rect.top + CharOffset, 'loyd');
  end;
end;

end.
```

Returns the amount required to repay a debt.

NUMBER : 2906
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : Returns the amount required to repay a debt.

PAYMENT()

Returns the periodic amount required to repay a debt.

```
function payment(princ, int, term: double): double;
var temp: double;
begin
  int := int / 100;
  temp := exp(ln(int + 1) * term);
  result := princ * ((int * temp) / (temp - 1));
end;
```

Syntax

PAYMENT(<principal expN>, <interest expN>, <term expN>)

<principal expN>

The original amount to be repaid over time.

<interest expN>

The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the term. It is to be expressed as a percentage. The number is divided by 100 inside the function.

<term expN>

The number of payments. Specify the term in the same time increment as the interest.

Description

Use PAYMENT( ) to calculate the periodic amount (payment) required to repay a loan or investment of <principal expN> amount in <term expN> payments. PAYMENT( ) returns a numeric value based on a fixed interest rate compounding over a fixed length of time. If <principal expN> is positive, PAYMENT( ) returns a positive number. If <principal expN> is negative, PAYMENT( ) returns a negative number. Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is 9.5 for payments made annually.

Express <interest expN> and <term expN> in the same time

increment. For example, if the payments are monthly, express the interest rate per month, and the number of payments in months. You would express an annual interest rate of 9.5%, for example, as 9.5/12, which is the 9.5% divided by 12 months. The formula used to calculate PAYMENT( ) is as follows:

$$pmt = princ * \frac{int*(1 + int)^{term}}{(1 + int)^{term} - 1}$$

where int = rate / 100 (as a percentage).

For the monthly payment required to repay a principal amount of $16860.68 in five years, at 9% interest, the formula expressed as a dBASE expression looks like this:

```
MyVar := PAYMENT(16860.68, 9/12, 60)          {Returns 350.00}
```

How to click and move components at runtime.

NUMBER : 2909
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1996

TITLE : How to click and move components at runtime.

Q: How can I program a component, such as a TPanel, so that I can move it around with a click and drag of the mouse?

A: This code goes on the OnMouseDown event of the component in question (a TPanel in this case):

```
procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
const
  SC_DragMove = $F012;   { a magic number }
begin
  ReleaseCapture;
  panel1.perform(WM_SysCommand, SC_DragMove, 0);
end;
```

Delphi 1.02 Maintenance Release Information

NUMBER : 2936
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : March 5, 1996

TITLE : Delphi 1.02 Maintenance Release Information

THE DELPHI 1.02 MAINTENANCE RELEASE

BRIEF OVERVIEW OF DELPHI 1.02

Thank you for your interest in Borland's new RADical application development tool, Delphi. Below you will find information regarding the product's overview as well as the problems fixed with this release, but here, we would like to give you a brief overview of Delphi 1.02.

- The Delphi 1.02 maintenance release is not a patch; it is a complete product.
- This release is available only through Borland. It is not available via CompuServe, Borland's Web site, Borland's FTP site, or resellers.
- This maintenance release includes the patch release of Delphi 1.01.
- Delphi 1.02 is not Delphi32. Delphi32 will be released as a separate product.

To order the Delphi 1.02 maintenance release, please call our Order Desk at 800-453-3375 extension 1327.

NOTE: All printed and online documentation assume that you have installed Delphi using the default directory structure.

TABLE OF CONTENTS
---

1. PRODUCT OVERVIEW

Delphi's visual design environment lets you create sophisticated Windows applications faster than any other development tool. Because Delphi is built around and optimizing native code compiler, Delphi applications are up to 10-20 times faster than interpreted code.

Delphi integrates the Borland Database Engine, so you have instant support for dBase, Paradox, and ODBC local databases.

Delphi includes the Local InterBase Server so you can create standalone client/server applications with a high-performance ANSI SQL-92 compatible database server.

Delphi includes Borland's award-winning ReportSmith report writing tool, which allows programmers to prepare innovative reports using live data in all popular database formats.

Delphi Client/Server includes all of Delphi plus high performance native drivers for Oracle, Sybase, Microsoft SQL Server, Informix and InterBase remote servers with unlimited application deployment. You also get team development support, ReportSmith SQL, a Visual Query Builder, source code to the Visual Component Library (VCL), and the Local InterBase Server Deployment Kit.

Delphi includes an Open Tools API capability that allows you to extend the Delphi environment to include your own tools, experts, and so on. By using this API, you can seamlessly integrate these extensions into the Delphi environment. For details, refer to the file TOOLINTF.PAS located in the \DELPHI\DOC for Delphi or in the \DELPHI\SOURCE\VCL directory for Delphi Client/Server. Source code for an example expert is also located in the \DELPHI\DEMOS\EXPERTS directory.

2. PROBLEMS FIXED IN THIS RELEASE

The purpose of this section is to provide a general list of problems that are fixed in the 1.02 Delphi release.

ReportSmith

- Create master/detail reports to combine multiple reports in one using heterogenous data.
- Choose to group, sort or summarize data locally or on a database server.
- Access data using Borland's BDE drivers.
- Include columns in a report for query-only or value-only.
- Utilize an updated ReportBasic macro language.
- Place page totals to display summary values for each page in a report.
- Create data dictionaries to use with the PC or SQL versions of ReportSmith, to simplify the view of the data.

Delphi

- Fixed problems in the Delphi online help system.

Local InterBase

- Improved performance of index creation and SQL request involving sort operations on data that exceeds the database cache as defined during database create.
- Improved I/O diagnostic and error messages from the server.
- Enhanced the Local InterBase engine to handle complex requests

that reference more than 500 columns causing internal buffers to exceed 64k.
- Improved performance of Local InterBase requests that reference more than 500 columns.
- Removed the Local InterBase internal requirement that databases have a page size of 1024. It is now possible to create and use a Database specifying any allowable InterBase page size.
- Numeric overflow errors are now trapped by Local InterBase and reported correctly to the client application.
- Fixed the incompatibility with Dashboard when using DashBoard, from Starfish Inc., and viewing users in the InterBase security database (Task|User Security) with the InterBase Server Manager tool.
- Fixed cleanup problems encountered when exiting the Windows Interactive SQL tool while executing a database validate (Task|Database Validate) in the InterBase Server Manager.

BDE

This version of the Borland Database Engine is the same version that is used with the new Visual dBASE 5.5.

- Fixed TQuery and TTable so that they support Oracle synonyms. Synonyms can now be viewed by typing its name in the name table property.
- Editing now supported of an opened Paradox table simultaneously in Delphi with referential integrity constraints.
- Fixed the ability to support Stored Procedures that have string parameters.
- Fixed problems relating to accessing tables on Lantastic 6.0 network.
- Removed the nine parameter limitation for SQL queries.
- Fixed problems relating to opening Access 2.0 files through ODBC.
- Fixed DBD so that it is now able to open Watcom 4.0 tables.
- Fixed 32K memory leak associated with live-query results sets and also for local queries involving joins.
- Fixed problem moving batch from ASCII fixed-length files.
- Now supports the SYBASE forcedindex feature.

3. DELPHI SUPPLEMENTAL DOCUMENTATION SET

The Supplemental Documentation Set includes the following:

- Object Pascal Language Reference Guide (objlang.pdf is 1.3M.)
- Delphi Visual Component Library Reference (vclref.pdf is 5.1M.)

In the United States, you can purchase hard copy versions of the Delphi Supplemental Documentation Set by calling the Borland order desk at 1-800-331-0877.

Internationally, you can purchase hard copy versions of the Delphi Supplemental Documentation Set by contacting the nearest Borland office.

4. MINIMUM SYSTEM REQUIREMENTS

Delphi requires Windows 3.1 or a 100% compatible operating system, an 80386 or newer processor (486 recommended), and 6Mb of system memory (Delphi Client/Server requires 8Mb, 12Mb or more is recommended for Client/Server development). A minimum installation requires approximately 30Mb of disk space (a full installation of Delphi Client/Server requires approximately 80Mb).

DCC.EXE, the DOS command-line compiler, requires at least 1Mb of extended memory.

Delphi has been tested under Windows 3.1, Windows for Workgroups 3.11, Windows NT 3.5, OS/2 Warp, and Windows 95.

Creating Dynamic Components at Runtime

NUMBER : 2938
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : June 24, 1996

TITLE : Creating Dynamic Components at Runtime

Using Delphi forms and components is simple. When coupled with the object inspector, controlling those objects requires little effort. Creating these types dynamically is also not difficult. This document is intended to give you some tips and hints on how to make dynamic component work for you.

(please note this term "dynamically" is subjective, behind the scenes, Delphi creates all objects dynamically. The information presented herein is for the programmer setting creation/properties/deletion the given type at run time)

All types (a form or a component) can be created dynamically. To do this, one needs to put a declaration in the VAR section of their code. This does not create an instance of the object, it creates a pointer. This pointer resides in the data segment (if the variable is declared globally) or the stack (if the variable is declared local to a procedure or function). In order to instantiate this class, you must call the constructor. This will allocate memory in the computers global heap for the class instance. Trying to access the component before allocating memory will produce a general protection fault.

The Create() constructor is a class method descended from the TObject Class. Create() returns a pointer. This method may or may not take one or more parameters. For most components (all objects which descend from TComponent are referred to as components), the constructor takes one parameter, the "owner" of type TComponent.

When dynamically creating a component, setting the Owner to "Self" is the most common practice. If you are in one of a form's methods, "Self" refers to that form in that context. If the owner is a valid object, freeing that object will also free the "owned" component. Another common parameter is "Application". This might be used for a visual component that will not be displayed to the program's users. However, most components do not require that you set a specific owner, so it is not uncommon to set the owner to Nil. Keep in mind, though, that you will not be able to change the owner afterwards. If you do pass Nil to a component's constructor, you must remember to call that component's Free method when you are through using the component.

After creation, but before they can be displayed, windowed components (those descending from TWinControl) require the Parent property to be set. At the time you set the Parent property, it's usually also a good time to set other properties of this components instance, including event handlers (ie, Width, Color, OnClick).

Event handlers are identical to those specified in the object inspector. Simply set the component's property name for the event you want to handle to name of the event handler method you want invoked. Example 1 below would call the method called "myclick" whenever the button is clicked. Please note this method will be sent the appropriate parameters, and its incoming parameter list must be exact.

Example 1:

```
var
  b1 : TButton;
begin
  .
  .
  .
  b1 := TButton.Create(Self);
  with b1 do begin
    Left := 20;
    Top := 20;
    Width := 90;
    Height := 50;
    Caption :=  my button';
    Parent := Form1;
    OnClick :=  MyClick;  { a procedure I defined somewhere else }
  end;
  .
  .
  .
end;
```

The next example demonstrates how to create a button at run time by clicking a predefined button. Note the different way the button has been created. Either way would work. Also note the buttons that are created are not freed in this code, they will be freed when the form is released.

```
unit Unit1;
interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  procedure myClick(Sender: TObject);
  end;

var
  Form1: TForm1;
```

```
const
  i : integer = 0;

implementation
{$R *.DFM}

procedure TForm1.myClick(Sender: TObject);
begin
  with Sender as TButton do
    Self.Caption := ClassName + ' ' + Name;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  with TButton.Create(self) do begin
    Left := 20;
    Top := 30 + i;
    Width := 120;
    Height := 40;
    Name := 'ThisButton' + IntToStr(i);
    Caption := 'There' + IntToStr(i);
    OnClick :=   MyClick;   { a procedure I defined somewhere else }
    Parent := Form1;
  end; {end with}
  inc(i, 40);
end; {end button1.click}

end.
```

Loading a Custom Cursor from a RES File

NUMBER : 2945
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : Loading a Custom Cursor from a RES File

Q: How do I use one of the cursor files in the c:\delphi\images\cursors?

A: Use the image editor to load the cursor into a RES file.
The following example assumes that you saved the cursor in the RES file as "cursor_1", and you save the RES file as MYFILE.RES.

```
(*** BEGIN CODE ***)
{$R c:\programs\delphi\MyFile.res}     { This is your RES file }

const PutTheCursorHere_Dude = 1;        { arbitrary positive number }

procedure stuff;
begin
   screen.cursors[PutTheCursorHere_Dude] := LoadCursor(hInstance,
                                                       PChar('cursor_1'));
   screen.cursor := PutTheCursorHere_Dude;
end;
```

Loading Bitmaps and Cursors from RES Files

NUMBER : 2947
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 5, 1996

TITLE : Loading Bitmaps and Cursors from RES Files

Loading Bitmaps and Cursors from RES files

Bitmaps and cursors can be stored in a resource (RES) files and linked into your application's EXE file. RES files can be created with Delphi's Image Editor or Borland's Resource Workshop that comes with the Delphi RAD Pack. Bitmaps and cursors stored in RES files (after being bound into an EXE or DLL) can be retrieved by using the API functions LoadBitmap and LoadCursor, respectively.

Loading Bitmaps

The LoadBitmap API call is defined as follows:

```
function LoadBitmap(Instance: THandle;
                    BitmapName: PChar): HBitmap;
```

The first parameter is the instance handle of the module (EXE or DLL) that contains the RES file you wish to get a resource from. Delphi provides the instance handle of the EXE running in the global variable called Hinstance. For this example it is assumed that the module that you are trying to load the bitmap from is your application. However, the module could be another EXE or DLL file. The following example loads a bitmap called BITMAP_1 from a RES file linked into the application's EXE:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bmp: TBitmap;
begin
  Bmp := TBitmap.Create;
  Bmp.Handle := LoadBitmap(HInstance,'BITMAP_1');
  Canvas.Draw(0, 0, Bmp);
  Bmp.Free;
end;
```

There is one drawback to using the LoadBitmap API call though LoadBitmap is a Windows 3.0 API call and loads in bitmaps only as DDBs (Device Dependent Bitmaps). This can cause color palette problems when retrieving DIBs (Device Independent Bitmaps) from RES files. The code listed below can be used to retrieve DIBs from RES files. This code loads the bitmap as a generic resource, puts it into a stream, and then does a LoadFromStream call which causes

Delphi to realize the color palette automatically.

```
procedure TForm1.Button1Click(Sender: TObject);
const
  BM = $4D42;   {Bitmap type identifier}
var
  Bmp: TBitmap;
  BMF: TBitmapFileHeader;
  HResInfo: THandle;
  MemHandle: THandle;
  Stream: TMemoryStream;
  ResPtr: PByte;
  ResSize: Longint;
begin
  BMF.bfType := BM;
  {Find, Load, and Lock the Resource containing BITMAP_1}
  HResInfo := FindResource(HInstance, 'BITMAP_1', RT_Bitmap);
  MemHandle := LoadResource(HInstance, HResInfo);
  ResPtr := LockResource(MemHandle);

  {Create a Memory stream, set its size, write out the bitmap
   header, and finally write out the Bitmap                    }
  Stream := TMemoryStream.Create;
  ResSize := SizeofResource(HInstance, HResInfo);
  Stream.SetSize(ResSize + SizeOf(BMF));
  Stream.Write(BMF, SizeOf(BMF));
  Stream.Write(ResPtr^, ResSize);

  {Free the resource and reset the stream to offset 0}
  FreeResource(MemHandle);
  Stream.Seek(0, 0);

  {Create the TBitmap and load the image from the MemoryStream}
  Bmp := TBitmap.Create;
  Bmp.LoadFromStream(Stream);
  Canvas.Draw(0, 0, Bmp);
  Bmp.Free;
  Stream.Free;
end;
```

Loading Cursors

The LoadCursor API call is defined as follows:

```
function LoadCursor(Instance: THandle;
                    CursorName: PChar): HCursor;
```

The first parameter is the Instance variable of the module that contains the RES file. As above, this example assumes that the module that you are trying to load the cursor from is your application. The second parameter is the name of the cursor.

Under the interface section declare:

```
const
```

```
  crMyCursor = 5; {Other units can use this constant}
```

Next, add the following two lines of code to the form's OnCreate event as follows:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Screen.Cursors[crMyCursor] := LoadCursor(HInstance, 'CURSOR_1');
  Cursor := crMyCursor;
end;
```

or you may want to change one of the standard Delphi cursors as follows (the Cursor constants can be found in the On-line Help under Cursors Property):

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {This example changes the SQL Hourglass cursor}
  Screen.Cursors[crSQLWait] := LoadCursor(HInstance, 'CURSOR_1');
end;
```

Note: Normally it is necessary to delete any cursor resources with the DeleteCursor, however, in Delphi this is not necessary because Delphi will delete the all cursors in the Cursors array.

SQL: Embedded Spaces in Field/Column Names

NUMBER : 2948
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : SQL: Embedded Spaces in Field/Column Names

Implementing SQL with spaces or special characters in field/column names

Implementing SQL statements in Delphi's TQuery component (or the SQL query facilities of Database Desktop, Visual dBASE or Paradox for Windows) requires special syntax for any columns that contain spaces or special characters.

Using the Biolife.DB table of from Delphi's demo data to illustrate, and without the use of any special syntax requirements, a SQL Select statement might be formed as follows,

```
SELECT
 Species No,
 Category,
 Common_Name,
 Species Name,
 Length (cm),
 Length_In,
 Notes,
 Graphic
FROM
 BIOLIFE
```

While appearing normal, the space in the species number and name columns and the column expressing length in centimeters - as well as the parentheses present - cause syntax errors.

Two changes must be taken to correct the syntax of the above SQL statement. First, any columns containing spaces or special characters must be surrounded by single (apostrophe) or double quotes. Secondly, a table reference and a period must precede the quoted column name. This second requirement is particularly important since a quoted string alone is interpreted as a string expression to be yielded as a column value. A properly formatted statement follows:

```
SELECT
 BIOLIFE."Species No",
 BIOLIFE."Category",
 BIOLIFE."Common_Name",
 BIOLIFE."Species Name",
 BIOLIFE."Length (cm)",
 BIOLIFE."Length_In",
 BIOLIFE."Notes",
 BIOLIFE."Graphic"
```

```
FROM
 "BIOLIFE.DB" BIOLIFE
```

The above example uses the table alias BIOLIFE as the table reference that precedes the column name. This reference may take the form of an alias name, the actual table name, or a quoted file name when using dBASE or Paradox tables. The following SQL statements would serve equally well.

Note: This SQL statement may be used provided that the necessary alias is already opened. In the case of the TQuery this means the alias is specified in the DatabaseName property.

```
SELECT
 BIOLIFE."Species No",
 BIOLIFE.Category,
 BIOLIFE.Common_Name,
 BIOLIFE."Species Name",
 BIOLIFE."Length (cm)",
 BIOLIFE.Length_In,
 BIOLIFE.Notes,
 BIOLIFE.Graphic
FROM
 BIOLIFE
```

If an alias is not available then the entire path to the table can be specified as in this example:

```
SELECT
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Species No",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Category",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Common_Name",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Species Name",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Length (cm)",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Length_In",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Notes",
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"."Graphic"
FROM
 "C:\DELPHI\DEMOS\DATA\BIOLIFE.DB"
```

Finally, two facilities that automatically handle this special formatting exist. The first is the Visual Query Builder that is a part of the Client/Server version of Delphi. The Visual Query Builder performs this formatting automatically as the query is built. The other facility is Database Desktop's Show SQL feature, available when creating or modifying a QBE-type query. After selecting Query|Show SQL from the main menu, the displayed SQL text may be cut and pasted where needed.

Dynamically Allocating Arrays

NUMBER : 2949
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : Dynamically Allocating Arrays

{
Dynamic Arrays

Is it possible to create a dynamically-sized array in Delphi?

Yes. First, you need to create an array type using the largest size you might possibly need. When creating a type, no memory is actually allocated. If you created a variable of that type, then the compiler will attempt to allocate the necessary memory for you. Instead, create a variable which is a pointer to that type. This causes the compiler to only allocate the four bytes needed for the pointer.

Before you can use the array, you need to allocate memory for it. By using AllocMem, you will be able to control exactly how many bytes are allocated. To determine the number of bytes you'll need to allocate, simply multiply the array size you want by the size of the individual array element. Keep in mind that the largest block that can be allocated at one time in a 16-bit environment is 64KB. The largest block that can be allocated at one time in a 32-bit environment is 4GB. To determine the maximum number of elements you can have in your particular array (in a 16-bit environment), divide 65,520 by the size of the individual element.
Example: 65520 div SizeOf(LongInt)

Example of declaring an array type and pointer:

```
type
  ElementType = LongInt;

const
  MaxArraySize = (65520 div SizeOf(ElementType));
    (* under a 16-bit environment *)

type
  MyArrayType = array[1..MaxArraySize] of ElementType;

var
  P: ^MyArrayType;

const
  ArraySizeIWant: Integer = 1500;
```

Then when you wish to allocate memory for the array, you could

use the following procedure:

```
procedure AllocateArray;
begin
  if ArraySizeIWant <= MaxArraySize then
    P := AllocMem(ArraySizeIWant * SizeOf(LongInt));
end;
```

Remember to make sure that the value of ArraySizeIWant is less than or equal to MaxArraySize.

Here is a procedure that will loop through the array and set a value for each member:

```
procedure AssignValues;
var
  I: Integer;
begin
  for I := 1 to ArraySizeIWant do
    P^[I] := I;
end;
```

Keep in mind that you must do your own range checking. If you have allocated an array with five members and you try to assign a value to the sixth member of the array, you will not receive an error message. However, you will get memory corruption.

Remember that you must always free up any memory that you allocate. Here is an example of how to dispose of this array:

```
procedure DeallocateArray;
begin
  P := AllocMem(ArraySizeIWant * SizeOf(LongInt));
end;
```

Below is an example of a dynamic array:

```
}

unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  ElementType = Integer;

const
  MaxArraySize = (65520 div SizeOf(ElementType));
    { in a 16-bit environment }

type
  { Create the array type.  Make sure that you set the range to
```

```
    be the largest number you would possibly need. }
  TDynamicArray = array[1..MaxArraySize] of ElementType;
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  { Create a variable of type pointer to your array type. }
  P: ^TDynamicArray;

const
  { This is a typed constant.   They are actually static
    variables hat are initialized at runtime to the value taken
    from the source code.   This means that you can use a typed
    constant just like you would use any other variable.   Plus
    you get the added bonus of being able to automatically
    initialize it's value. }
  DynamicArraySizeNeeded: Integer = 10;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Allocate memory for your array.   Be very careful that you
    allocate the amount that you need.   If you try to write
    beyond the amount that you've allocated, the compiler will
    let you do it.   You'll just get data corruption. }
  DynamicArraySizeNeeded := 500;
  P := AllocMem(DynamicArraySizeNeeded * SizeOf(Integer));
  { How to assign a value to the fifth member of the array. }
  P^[5] := 68;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  { Displaying the data. }
  Button1.Caption := IntToStr(P^[5]);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  { Free the memory you allocated for the array. }
  FreeMem(P, DynamicArraySizeNeeded * SizeOf(Integer));
end;

end.
```

Resource Expert: What It Is and How to Install It

NUMBER : 2950
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : March 4, 1996

TITLE : Resource Expert: What It Is and How to Install It

Resource Expert: what it is and how to install and use it.

What is the Resource Expert

The Resource Expert is a Delphi expert add-on that is available as a part of the Delphi Rad Pack. The expert assists the programmer in porting existing projects to Delphi by converting dialog and menu resource scripts intended for use in traditional Windows applications. Dialog resources and their contents are converted to Delphi forms with the analogous controls converted to Delphi components.

How to install the Resource Expert

The Resource Expert is installed via the Delphi Rad Pack's Resource Workshop 4.5 install procedure. Once installed, it is incorporated into the Delphi component library and is available as an option on the Delphi Help Menu or via the Experts page of the Forms Gallery dialog. Installation of the Resource Expert files may be installed from within the Windows environment or from the command line under Windows 95 or Windows NT.

To install the Resource Expert files from within Windows,

1) Begin the installation procedure for Borland Resource Workshop.
2) On the third dialog, entitled 'Resource Workshop - Resource Expert Options', ensure that the 'Install Resource Expert' check box is checked.
3) The 'Install to:' entry indicates the destination directory for the Resource Expert files, indicating C:\DELPHI\RCEXPERT by default. Change this entry as needed.
4) Proceed with the rest of the Resource Workshop installation process as normal.

To install the Resource Expert files from the command line, type the following commands,

1) MD C:\DELPHI\RCEXPERT
2) CD C:\DELPHI\RCEXPERT
3) E:\INSTALL\RW\UNPAQ -X E:\INSTALL\RW\RESEXP.PAK

Note: The last command above assumes that the E: drive is a CD-ROM drive containing the Rad Pack Installation CD.

Once the installation of the Resource Expert files is completed, the Delphi Component Library must be recompiled. To do this,

1) Load Delphi.
2) Select Options|Install Components.
3) Click the Add... button.
4) When the Add Module dialog appears, enter the full path name of the rcexpert.pas file or find the file via the Browse... button.
5) Finally, choose the OK button on the Install Components Dialog.

How to use the Resource Expert

To convert a resource script, all source files normally required to compile the script must be present. This would include .RC, .MNU, or .DLG file(s) and any .H or .PAS include files they refer to. Resource scripts typically use WINDOWS.H and BWCC.H. These files are usually located in directories such as \BC4\INCLUDE or \BP7\UNITS. The Resource Expert supports the RC language extensions defined by Resource Workshop.

Again, the Resource Expert may be invoked via the Help|Resource Expert menu option or via the Experts page of the Forms Gallery dialog. The latter will appear if the 'Use on new form' check box is checked on the Preferences page of the Environment Options dialog.

Once the Resource Expert has been invoked, click the 'Next' button to bypass the page that introduces the expert to the user. The second page of the expert allows the user to select the resource scripts to convert. A number of scripts may be chosen provided that they all reside in the same directory. The particular type of script to view (.RC, .DLG or .MNU) can be selected via the 'List Files of Type' combo box. After selecting the scripts to convert, click the 'Next' button again. The third page presents a single 'Include Path' edit box. Enter the list of directories containing .H, .INC, or .PAS include files used by the resource scripts, (if any). Each directory name should be separated by a semicolon. Again, click the 'Next' button to continue. On the fourth and final page of the expert, the 'Convert' button appears. Clicking it begins the actual conversion process. If the resources script contain many dialogs, the 'Show all forms' check box may be un-checked in order to speed the conversion process and to minimize impact on Windows system resources.

If a syntax error is encountered during the conversion process, the erroneous statement will be discarded and conversion will

resume at the next statement or block. Errors will be noted in the log file ERRLOG.TXT and displayed in a Delphi editor window.

Once the conversion process is complete, separate forms for each dialog resource will have been created. For menu resources, a simple form containing the converted menu component will have been created. If a project was active before the conversion began, the converted forms are added to the project. Each form may now be used and modified as would any Delphi form.

Delphi Configuration Files

NUMBER : 2951
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : March 4, 1996

TITLE : Delphi Configuration Files

Delphi employs a number of files for its configuration, some global to the Delphi environment, some project specific. Chief among all the configuration files is DELPHI.INI. DELPHI.INI resides in the Windows directory and contains most of the configurable items to be found within Delphi. Being the largest Delphi configuration file, DELPHI.INI contains settings to control the Delphi speed bar, component palette, component library, gallery, installed experts, editor, printing, tools menu and all the environment options found on the Environment Options dialog. This document will explore DELPHI.INI in depth.

DELPHI.CBT is a copy of DELPHI.INI (minus the ReportSmith section) that is installed along with Delphi and may serve as a sort of backup for restoring the original Delphi setup. It resides in the Windows directory along with DELPHI.INI. Below is the ReportSmith section and its one entry that should be placed in a DELPHI.INI created from DELPHI.CBT.

```
[ReportSmith]
ExePath=C:\RPTSMITH
```

Delphi also makes use of Desktop (.DSK) files. Desktop files, like most Delphi configuration files, are formatted in the same manner as .INI files, with section headers and individual settings in each section. The purpose of desktop files is to retain the appearance and content of the Delphi desktop between sessions or between projects. Each desktop file contains information regarding the presence and appearance of the Delphi main window, the Object Inspector, the Alignment Palette, the Project Manager, and the Watch, Breakpoint, CallStack, and component list windows. Also kept in each desktop file is the number of editor windows open as well as the names, number and order of files open in each editor window.

If the 'Desktop files' check box (on the Preferences page of the Environment Options dialog) is checked, Delphi will automatically create desktop files for each project closed and saved. Each desktop file carries the same root name as the saved project file. If no project is active when Delphi exits, a default desktop file, DELPHI.DSK, is created. The last active project determines which desktop file Delphi loads at startup. Again, if no project was active when Delphi exited last, then DELPHI.DSK is loaded. While project specific desktop files reside in the same directory with the corresponding project, DELPHI.DSK resides

in the \DELPHI\BIN directory. The PrivateDir setting in the Globals section of DELPHI.INI may be used to relocate DELPHI.DSK to a different location.

Option files (.OPT) are another INI-like file in which Delphi maintains values directly corresponding to those settings on the Compiler, Linker, and Directories/Conditionals pages of the Project Options dialog. Each of these pages has a corresponding section in the option file and each setting has a individual entry in that section. Each option file also retains the last parameter string entered via the Run Parameters dialog. An option file is created for each project saved. Like .DSK files, the root name of the .OPT file is the same as its corresponding project and reside in the same directory as that project.

A default option file, DEFPROJ.OPT, is created if the Default check box of the Project Options dialog is checked. The settings in DEFPROJ.OPT serve as the default project settings each time a new project is created.

Additionally, the Delphi command line compiler, DCC.EXE, supports the use of the configuration file DCC.CFG. DCC.CFG is a text file opened when the command line compiler starts and is used in addition to options entered on the command line. Command line options may be placed in DCC.CFG, each on a separate line. When DCC starts, it looks for DCC.CFG in the current directory. If it is not found there, the directory in which DCC.EXE resides is then searched. A sample DCC.CFG follows:

```
/b
/q
/v
/eC:\DELPHI\WORK
```

The above settings instruct the command line compiler to build all units (/b), compile without displaying file names and line numbers (/q), append debug information to the .EXE (/v), and place the compiled units and exEcutable in the C:\DELPHI\WORK directory (/eC:\DELPHI\WORK). The contents of the installed DCC.CFG are included below to serve in restoring it should it be deleted or damaged.

```
/m
/cw
/rD:\DELPHI\LIB
/uD:\DELPHI\LIB
/iD:\DELPHI\LIB
```

STDVCS.CFG is a file installed with the Client/Server of Delphi, but is only used in conjunction with the Version Control manager DLL. The contents of the installed STDVCS.CFG are included here to serve in restoring it should it be deleted or damaged.

```
NODELETEWORK WRITEPROTECT
NOCASE VCSID

COMMENTPREFIX .PAS = "{ "
COMMENTPREFIX .PRJ = "{ "

NOEXPANDKEYWORDS .FRM
NOEXPANDKEYWORDS .EXE
NOEXPANDKEYWORDS .DLL
NOEXPANDKEYWORDS .DOC
NOEXPANDKEYWORDS .ICO
NOEXPANDKEYWORDS .BMP
```

Lastly, MULTIHLP.INI is a file Delphi uses to provide context-sensitive help across multiple help files. This file should not be modified; doing so may cause the Delphi Help system to behave erratically. The contents of the installed MULTIHLP.INI are included here to serve in restoring it should it be deleted or damaged.

```
[Index Path]
DELPHI.HLP=C:\DELPHI\BIN
WINAPI.HLP=C:\DELPHI\BIN
CWG.HLP=C:\DELPHI\BIN
CWH.HLP=C:\DELPHI\BIN
LOCALSQL.HLP=C:\DELPHI\BIN
VQB.HLP=C:\DELPHI\BIN
SQLREF.HLP=C:\IBLOCAL\BIN
WISQL.HLP=C:\IBLOCAL\BIN
BDECFG.HLP=C:\IDAPI
RPTSMITH.HLP=C:\RPTSMITH
RS_DD.HLP=C:\RPTSMITH
SBL.HLP=C:\RPTSMITH
RS_RUN.HLP=C:\RPTSMITH
DBD.HLP=C:\DBD
```

Note:

What follows below is a comprehensive dissection of the DELPHI.INI file. In order to save space, a few conventions were observed in the describing possible values for settings.

Where only one of a limited set of values is applicable, a pipe symbol is used to separate each of the possible value, e.g.:

```
MapFile=0|1|2|3
```

allows only the values 0, 1, 2, or 3

Where a single value within a range is applicable, the range of values is presented inside brackets with the minimum and maximum values separated by two periods, e.g.:

```
GridSizeX=[2..128]
```

permits any value between 2 and 128, inclusively.

==========

Section: [Globals] - The Globals section contains settings not included in other sections and that have an effect on Delphi as a whole. Items in the Globals section may be changed only by editing DELPHI.INI.

PrivateDir=

This item controls where Delphi both creates and locates the files DELPHI.DSK, DELPHI.DMT, DEFPROJ.OPT and STDVCS.CFG. The default location is the \DELPHI\BIN directory. If Delphi is run from a read-only directory (or from a CD-ROM) this item should be set to a writeable directory, either on a network or local drive. This item should contain a fully qualified path, including the drive letter. Example:

```
PrivateDir=J:\USERS\JSMITH     ; Private network directory
```

HintColor=

This item controls the color of the fly-by hint window for the Delphi IDE. The value may be a decimal or hex constant, or one of the symbolic color constants defined in VCL (e.g. clCyan). Note that the text in the hint window is always painted using clWindowText. The default value is clYellow.

PropValueColor=

This item controls the color of the text in the right-hand (value) pane of the Object Inspector. The value may be a decimal or hex constant, or one of the symbolic color constants defined in VCL (e.g. clBlue). The default value is clWindowText.

Section: [Library] - The Library section contains entries for those settings found on the Library page of the Environment Options dialog (accessed via Options|Environment). The options in this section take effect when the Options|Rebuild Library menu option is chosen.

SearchPath=

Specifies search paths where the compiler can find the units needed to build the component library. Path names should be listed consecutively, separated by a semicolon. This entry is

changed via the 'Library Path' combo box. Example:

SearchPath=D:\DELPHI\LIB;d:\delphi\rcexpert

ComponentLibrary=

Specifies the name of the active component library. This item is changed via the Options|Open Library menu option. It may also be changed from the 'Library filename' edit of the Install Components dialog (accessed via Options|Install Components). Example:

ComponentLibrary=D:\DELPHI\BIN\REXPERT.DCL

SaveLibrarySource=0|1

Indicates whether Delphi saves the source code for the component library when installing new components or rebuilding it via Options|Rebuild Library. A setting of 1 causes the project source to be saved using the library file's root name with a .DPR extension. The default value is 0. This setting is changed via the 'Save library source code' check box.

MapFile=0|1|2|3

Determines the type of map file produced, if any, when the component library is rebuilt. The map file is placed in the same directory as the library, and it has a .MAP extension. The default value is 0. This setting is changed via the 'Map file' radio button group.

Option	Effect
0 - Off	Does not produce map file.
1 - Segments	Linker produces a map file that includes a list of segments, the program start address, and any warning or error messages produced during the link.
2 - Publics	Linker produces a map file that includes a list of segments, the program start address, any warning or error messages produced during the link, and a list of alphabetically sorted public symbols.
3 - Detailed	Linker produces a map file that includes a list of segments, the program start address, any warning or error messages produced during the link, a list of alphabetically sorted public symbols, and an additional detailed segment map. The detailed segment map includes the address, length in bytes, segment name, group, and module information.

LinkBuffer=0|1

Specifies the location of the link buffer. A setting of 1 causes Delphi to use available disk space for the link buffer; 0 causes the use of available memory. The default value is 0. This setting is changed via the 'Link Buffer' radio button group.

DebugInfo=0|1

Determines whether the component library file is compiled and linked with debug information. A setting of 1 causes the inclusion of debug information. The default setting is 0. The setting is changed via the 'Compile with debug info' check box.

Section: [Gallery] - The Gallery section controls the use and base location of the form and project galleries. It contains those settings found in the Gallery: group box on the Preferences page of the Environment Options dialog.

BaseDir=

Points to the directory where Delphi attempts to find Gallery files. To share a gallery directory with other users, set this item to point to a shared network directory. This item should contain a fully qualified path, including the drive letter. This entry may be changed only by editing DELPHI.INI. Example:

```
BaseDir=D:\DELPHI\GALLERY
```

GalleryProjects=0|1

Indicates whether Delphi displays the Browse Gallery dialog box when the File|New Project menu option is chosen. A setting of 1 causes the Browse Gallery dialog box to display. The default setting is 0. The setting is changed via the 'Use on New Project' check box.

GalleryForms=0|1

Indicates whether Delphi displays the Browse Gallery dialog box when the File|New Form menu option is chosen. A setting of 0 prevents the Browse Gallery dialog box from displaying. The default setting is 1. The setting is changed via the 'Use on New Form' check box.

Section: [Experts] - The Experts section lists the Experts which Delphi will attempt to load and initialize upon startup. Any value may be used on the left of the equals sign, as the item name is not interpreted. Borland recommends using a combination of the vendor name and the product name. Example:

```
[Experts]
ComponentWare.CommExpert=c:\delphi\cware\commexpt.dll
CodeFast.TheExpert=c:\delphi\codefast\codefast.dll
```

Section: [ReportSmith] - The ReportSmith section contains just one entry which specifies the directory in which ReportSmith is installed.

ExePath=

ExePath indicates the location of RPTSMITH.EXE. This entry is placed in DELPHI.INI at install time and may be changed only by editing DELPHI.INI. Example:

```
ExePath=D:\RPTSMITH
```

Section: [Session] - The Session section and its one entry identify the active project when Delphi was last closed.

Project=

Identifies the active project when Delphi was last closed. This setting is only meaningful if the DesktopFile setting in the AutoSave section is set to 1. This setting also serves to identify the project's desktop file (using a .DSK extension). This setting is updated automatically when Delphi exits. Example:

```
Project=D:\DELPHI\WORK\MAILAPP.DPR
```

Section: [MainWindow] - The MainWindow section defines characteristics of the Delphi main window as they relate to the speedbar and component palette. The SpeedBar Layout section details the actual contents of the speedbar. Likewise, the <libraryname>.Palette section details the actual contents of the component palette.

Split=[-1..400]

Indicates the horizontal position if the vertical bar separating the speedbar and component palette. The default value is 183. This setting is changed by moving the split bar with the mouse.

SpeedHints=0|1

Determines whether hints are displayed as the mouse passes over buttons on the speedbar. A setting of 0 prevents the display of speedbar hints. The default setting is 1. This setting is changed using the Show Hints menu option of the speedbar speedmenu.

PaletteHints=0|1

Determines whether hints are displayed as the mouse passes over buttons on the palette. A setting of 0 prevents the display of palette hinsts. The default setting is 1. This setting is changed using the Show Hints menu option of the palette speedmenu.

Speedbar=0|1

When set to 0, prevents the display of the speedbar. The default setting is 1. This setting is changed via the View|Speedbar menu option or via the Hide option of the speedbar speedmenu.

Palette=0|1

When set to 0, prevents the display of the component palette. The default setting is 1. This setting is changed via the View|Component Palette menu option or via the Hide option of the component palette speedmenu.

Section: [Speedbar Layout] - The Speedbar Layout details the specific contents of the speedbar. The contents of this section are changed via the Configure option of the speedbar speedmenu.

Count=[0..52]

Specifies the number of buttons on the speedbar. The default is 14.

Button[0..51]=n,x,y

This entry appears once for each button on the speedbar. Each

button entry is uniquely numbered, the first being Button0. The number n identifies a unique pre-defined id code. The x value is a number specifying the horizontal position of the button on the speedbar. The y value is a number specifying the vertical position of the button on the speedbar. Below is a listing of the default speedbutton set and their corresponding menu options.

```
Button0=30001,4,2          ; File|Open Project...
Button1=30002,27,2         ; File|Save Project
Button2=30007,4,25         ; File|Open File...
Button3=30008,27,25        ; File|Save File
Button4=30009,50,2         ; File|Add File...
Button5=30010,50,25        ; File|Remove File...
Button6=30069,79,2         ; View|Units...
Button7=30070,102,2        ; View|Forms...
Button8=30068,79,25        ; View|Toggle Form/Unit
Button9=30004,102,25       ; File|New Form
Button10=30090,131,2       ; Run|Run
Button11=30093,154,2       ; Run|Program Pause
Button12=30092,131,25      ; Run|Trace Into
Button13=30091,154,25      ; Run|Step Over
```

Section: [Desktop] - The Desktop section contains a single entry that determines which desktop settings are saved when Delphi exits. This section and its one entry is only meaningful if the DesktopFile entry in the AutoSave section is 1.

SaveSymbols=0|1

Determines if browser symbol information is saved along with Desktop information when Delphi exits. This setting is changed via the 'Desktop contents:' radio button group box. The default setting is 1.

Option	Effect
0 - Desktop only	Saves directory information, open files in the editor, and open windows.
1 - Desktop and symbols	Saves desktop information and browser symbol information from the last successful compile.

Section: [AutoSave] - The Autosave section determines which files and options are saved automatically when the current project is run or when Delphi exits. This section corresponds to the 'Autosave options:' group box of the Preferences page of the Environment Options Dialog.

EditorFiles=0|1

When set to 1, causes Delphi to save all modified files in the Code Editor when Run|Run, Run|Trace Into, Run|Step Over, or Run|Run To Cursor are chosen, or when Delphi exits. The default setting is 0. This setting is changed via the 'Editor files' check box on the Preferences page of the Environment Options Dialog.

DesktopFile=0|1

When set to 0, prevents Delphi from saving the arrangement of the desktop when a project is closed or when Delphi exits. The default setting is 1. This setting is changed via the 'Desktop' check box on the Preferences page of the Environment Options Dialog.

Note: Further discussion regarding desktop files are discussed below under Desktop (.DSK) files.

Section: [FormDesign] - The FormDesgin section contains those settings that control the appearance and behavior of a forms grid at design time. This section corresponds to the 'Form designer:' group box of the Preferences page of the Environment Options Dialog.

DisplayGrid=0|1

Determines the design time visibility of the dots that comprise the form grid. A setting of 0 avoids grid display. The default setting is 1. This setting is changed via the 'Display grid' check box.

SnapToGrid=0|1

Indicates whether components are automatically aligned with the grid when components are moved with the mouse. A setting of 0 avoids grid alignment. The default setting is 1. This setting is changed via the 'Snap to grid' check box.

GridSizeX=[2..128]

Sets grid spacing in pixels along the x-axis. The default value is 8. This setting is changed via the 'Grid Size X' edit.

GridSizeY=[2..128]

Sets grid spacing in pixels along the y-axis. The default

value is 8. This setting is changed via the 'Grid Size Y' edit.

DefaultFont=

This item controls the default font for new forms. The name of the font, the font size, and optionally the style of the font may be entered, each separated by commas. (Supported font styles are "bold" and "italic.") This setting may be changed only by editing DELPHI.INI. Example:

DefaultFont=MS Sans Serif, 8, bold, italic

Section: [Debugging] - The Debugging section contains those settings that control integrated debugging and the appearance of Delphi during project execution. This section corresponds to the 'Debugging:' group box of the Preferences page of the Environment Options Dialog.

IntegratedDebugging=0|1

Allows or prevents the uses of the Delphi Integrated Debugger. A setting of 0 prevents integrated debugging. The default setting is 1. This setting is changed via the 'Integrated Debugging' check box.

DebugMainBlock=0|1

When set to 1, causes the debugger to stop at the first unit initialization that contains debug information. The default setting is 0. This setting is changed via the 'Step program block' check box.

BreakOnExceptions=0|1

When set to 1, stops the application when an exception is encountered and displays the following the exception class, exception message and the location of the exception. When set to 0, exceptions do not stop the running application. The default setting is 1. This setting is changed via the 'Break on exception' check box.

MinimizeOnRun=0|1

When set to 1, minimizes Delphi when the current project is executed. The default is 0. This setting is changed via the 'Minimize on run' check box.

HideDesigners=0|1

When set to 1, hides designer windows, such as the Object Inspector and Form window, while the application is running. The default setting is 1. This setting is changed via the 'Hide designers on run' check box.

NoResetWarning=0|1

When set to 1, prevents Delphi from presenting a warning message when Program Reset is selected. The default setting is 0. This setting may be changed only by editing DELPHI.INI.

Section: [Compiling] - The compiling section contains a single entry that determines whether the user is presented with a dialog that reports compiler progress. This section corresponds to the 'Compiling:' group box of the Preferences page of the Environment Options Dialog.

ShowCompilerProgress=0|1

Specifies whether compilation progress is reported. A setting of 1 causes Delphi to display a window detailing compilation progress. The default setting is 0. This setting is changed via the 'Show compiler progress' check box.

Section: [Browser] - The Browser section contains settings that are found on the Browser page of the Environment Options dialog. These settings specify how ObjectBrowser functions and what symbol information is displayed.

Filters=

This setting determines which filters are active in the Object Browser. The value is the sum of the values listed below for each filter desired.

Value	Filter
2	Constants
4	Types
8	Variables
16	Functions and Procedures
32	Properties
128	Inherited
256	Virtuals only
1024	Private
2048	Protected

4096	Public
8192	Published

The default setting is 15806, which activates all filters. Each filter corresponds to a check box in the 'Symbol filters:' group box. For example, the following setting activates the Properties, Public and Published filters:

```
Filters=12320   ; 8192 + 4096 + 32 = 12320
```

InitialView=1|2|3

InitialView determines the type of information the browser displays when first opened. The default setting is 2. This setting is changed via the 'Initial view:' radio button group box.

Value	Viewed
1	Units
2	Objects
3	Globals

Sort=0|1

When set to 1, causes Delphi to display symbols in alphabetical order by symbol name. When set to 0, symbols display in order of declaration. The default setting is 0. This setting is changed via the 'Sort always' check box.

QualifiedSymbols=0|1

When set to 1, causes Delphi to display the qualified identifier for a symbol. When set to 0, only the symbol name is displayed. The default setting is 0. This setting is changed via the 'Qualified symbols' check box.

CollapsedNodes=

Specifies which branches of the object tree hierarchy are collapsed when the ObjectBrowser is started. This entry is a list of class names, separated by separated by semicolons. This setting is changed via the 'Collapse Nodes:' combo box. Example:

```
CollapsedNodes=Exception;TComponent
```

ShowHints=0|1

Determines whether hints are displayed as the mouse passes over filter buttons. A setting of 0 prevents the display of filter

hints. The default setting is 1. This setting is changed using the Show Hints menu option of the ObjectBrowser speedmenu.

Section: [Custom Colors] - The Custom colors section lists up to sixteen user defined colors. Each color is specified as a six-digit hexadecimal RGB value. An unused color entry is indicated by the hexadecimal value FFFFFFFF. Entries in this section are created and updated via the Color dialog of any components Color property (accessed by double-clicking the entry area of the Color property).

Color[A..P]=

Specifies an individual RGB value for a user defined color.

Section: [Print Selection] - The Print Selection section contains those options that appear when the File|Print menu option is chosen. These settings correspond to the options displayed in the 'Options:' group box.

HeaderPage=0|1

When set to 1, Delphi includes the name of the file, current date, and page number at the top of each page. The default setting is 0. This setting is changed via the 'Header/page number' check box.

LineNumbers=0|1

When set to 1, Delphi places line numbers in the left margin of the printed output. The default setting is 0. This setting is changed via the 'Line numbers' check box.

SyntaxPrinting=0|1

When set to 1, Delphi uses bold, italic, and underline characters to indicate elements with syntax highlighting. When set to 0, Delphi uses no special formatting when printing. The default value is 1. This setting is changed via the 'Syntax print' check box.

UseColor=0|1

When set to 1, causes Delphi to print colors that match colors on screen. This option requires that the current printer

support color. The default value is 0. This setting is changed via the 'Use Color' check box.

WrapLines=0|1

When set to 1, causes Delphi to use multiple lines to print characters beyond the page width. When set to 0, code lines are truncated and characters beyond the page width do not print. The default value is 0. This setting is changed via the 'wrap lines' check box.

LeftMargin=[0..79]

Specifies the number of character spaces used as a margin between the left edge of the page and the beginning of each line. The default value is 0. This setting is changed via the 'Left margin' edit.

Section: [Highlight] - The Highlight section contain those settings that determine the syntax and context specific colors used in the Code Editor. The settings in this section are changed via the Editor Colors page of the Environment Options dialog.

ColorSpeedSetting=0|1|2|3

Determines which color scheme was last selected. Changing this setting directly does not affect the actual colors used for individual elements. The Color SpeedSetting combo box does not save color schemes; it only serves as a quick means of setting all color elements at once. The default setting is 0. The table below shows each value's corresponding speedsetting.

Value	SpeedSetting
0	Defaults
1	Classic
2	Twilight
3	Ocean

<Element color>=

All the color entries correspond to a single color element. Each color element entry uses the following format:

<Element name>=fRGB,bRGB,attr,deffore,defback,fcell,bcell

Value code	Meaning
fRGB	Foreground RGB value

bRGB	Background RGB value
attr	Text attribute; zero or more of B, I and U
deffore	Use default foreground color (1=yes, 0=no)
defback	Use default background color (1=yes, 0=no)
fcell	Foreground color grid cell number
bcell	Background color grid cell number

Section: [Editor] - This section describes the appearance and behavior of the Delphi Code Editor. Settings from both the Editor options and Editor display pages are detailed here.

DefaultWidth=
DefaultHeight=

These two items, if present, control the initial width and height of the Delphi Code Editor window. Delphi does not update these values, but it does read them each time a Code Editor is created. The default width is 406; the default height is 234. These settings may be changed only by editing DELPHI.INI.

FontName=
FontSize=

These settings specify the name and size, respectively, of a mono-spaced font that the Code Editor uses to display text. Courier New is the default font, 10 the default size. These entries may be changed via the 'Editor font:' and 'Size:' combo boxes on the Editor display page.

BlockIndent=[1..16]

Specifies the number of spaces to indent a marked block. The default value is 1. This setting may be changed via the 'Block indent' combo box on the Editor display page.

UndoLimit=[0..]

Specifies the number of keystrokes that can be undone, which is limited by available memory. The default value is 32,767. This setting may be changed via the 'Undo limit:' combo box on the Editor Options page.

TabRack=

Determines the columns at which the cursor will move to each time the Tab key is pressed. Each successive tab stop must be separated by a space and must be larger than its predecessor.

If only one number is specified, tab stops are spaced apart evenly, using that number. If two numbers are specified then tab stops occur at the specified positions and at positions that mark the difference between the two values. The default tab stops are 9 and 17. This setting may be changed via the 'Tab stops:' combo box on the Editor Options page. Note: this option has no effect if the smart tabs setting is enabled.

RightMargin=[0..1024]

Specifies the right margin of the Code Editor. The default value is 80. The valid range is 0 to 1024. This setting may be changed via the 'Right margin:' combo box on the Editor display page.

Extensions=

Combo Box
Specifies file masks of those files that will display with syntax highlighting. Typically, only specific extensions are included. The default setting is '*.PAS;*.DPR;*.DFM;*.INC;*.INT'. This setting may be changed via the 'Syntax extensions:' combo box on the Editor Options page. Example:

Extensions=*.PAS;*.DPR;*.SRC

FindTextAtCursor=0|1

When set to 1, causes Delphi to Place the text at the cursor into the 'Text To Find' combo box in the Find Text dialog box when the Search|Find menu option is chosen. When set to 0, the default setting, the search text must be typed in. This entry may be changed via the 'Find text at cursor' check box on the Editor Options page.

BRIEFRegularExpressions=0|1

When set to 1, permits the use of Brief-style regular expressions when searching for text. The default setting is 0. This entry may be changed via the 'BRIEF regular expressions' check box on the Editor Options page.

PreserveLineEnds=0|1

Determines whether end-of-line characters are changed to carriage return/line feed pairs or are preserved. When set to 0, Delphi converts end-of-line characters to carriage return/line feed pairs. The default value is 1. This entry may be changed via the 'Preserve Line Ends' check box on the Editor display page.

FullZoom=0|1

Determines whether the Code Editor fills the entire screen when maximized. When set to 0 (the default), the Code Editor does not cover the Delphi main window when maximized. A setting of 1 allows the Code Editor window to encompass the entire screen. This setting may be changed via the 'Zoom to full screen' check box on the Editor Display page.

DoubleClickLine=0|1

When set to 1, causes Delphi to highlight the whole line when the user double-clicks any character in the line. When set to 0 (the default), only the selected word is highlighted. This entry may be changed via the 'Double click line' check box on the Editor Options page.

BRIEFCursors=0|1

Determines whether Delphi uses BRIEF-style cursor shapes in the Code Editor. A setting of 1 causes Delphi to use Brief-style cursors. The default setting is 0. This setting may be changed via the 'BRIEF cursor shapes' check box on the Editor Display page.

ForceCutCopyEnabled=0|1

When set to 1, enables the Edit|Cut and Edit|Copy menu options, even when no text is selected. The default setting is 0. This entry may be changed via the 'Force cut and copy enabled' check box on the Editor Options page.

KeyBindingSet=0|1|2|3

Determines which pre-defined key mapping set Delphi recognizes. The default setting is 0. This setting may be changed via the 'Keystroke mapping:' list box on the Editor Display page. The table below identifies the appropriate mapping for the desired value.

Value	Mapping
0	Default
1	Classic
2	Brief
3	Epsilon

Mode=

This setting determines the state of sixteen of the options available on the Editor Options page and two of the options on the Editor Display page. The value is the sum of the values listed below for each check box checked. Unless noted, all the options below correspond to a similarly named check box on the Editor Options page.

- 1 Insert mode - Inserts text at the cursor without overwriting existing text.
- 2 Auto indent mode - Positions the cursor under the first nonblank character of the preceding nonblank line when Enter is pressed.
- 4 Use tab character - Inserts tab character. If disabled, inserts space characters. This option and the Smart Tabs option are mutually exclusive. enabled, this option is off.
- 16 Backspace un-indents - Aligns the insertion point to the previous indentation level (out-dents it) when Backspace is pressed, if the cursor is on the first nonblank character of a line.
- 32 Keep trailing blanks - Saves trailing spaces and tabs present at the end of a line.
- 64 Optimal fill - Begins every auto-indented line with the minimum number of characters possible, using tabs and spaces as necessary.
- 128 Cursor through tabs - Enables the arrow keys to move the cursor to the beginning of each tab.
- 256 Group undo - Undoes the last editing command as well as any subsequent editing commands of the same type when Alt+Backspace, Ctrl+Z is pressed or the Edit|Undo menu option is chosen.
- 512 Persistent blocks - Keeps marked blocks selected even when the cursor is moved, until a new block is selected.
- 1024 Overwrite blocks - Replaces a marked block of text with whatever is typed next. If Persistent Blocks is also selected, text entered is added to the currently selected block.
- 4096 Create backup file - Creates a backup file when source files are saved. This item is set via the 'Create backup file' check box on the Editor Display page.
- 8192 Use Syntax highlight - Enables syntax highlighting.
- 16384 Visible right margin - Enables the display of a line at the right margin of the Code Editor. This item is set via the 'Visible right margin' check box on the Editor Display page.
- 32768 Smart tabs - Tabs to the first non-whitespace character in the preceding line. This option and the Smart Tabs option are mutually exclusive.
- 131072 Cursor beyond EOF - Allows cursor positioning beyond the end-of-file.
- 262144 Undo after save - Allows retrieval of changes after a save.

EditorSpeedSetting=0|1|2|3

Determines which editor emulation scheme was last selected. Changing this setting directly does not affect the actual keystroke mapping or the editor options used. The Editor SpeedSetting combo box does not save emulation schemes; it only serves as a quick means of setting many editor options at once. The default setting is 0. The table below shows each value's corresponding speedsetting.

Value	SpeedSetting
0	Default keymapping
1	IDE classic
2	Brief emulation
3	Epsilon emulation

Section: [<Library name>.Palette] - This section describes the content of the Component Palette. Each entry name in this section matches a single page name on the component palette. The value for each entry is a list of the component type names that appear on that page, each separated by a semicolon. This section appears once for each component library configured via the Palette page of the Environment Options dialog.

Section: [Transfer] - The Transfer section defines those items that appear on the Tools menu. Entries in this section are defined when using the Tool Properties dialog. The Tool Properties dialog is itself accessed via the Options|Tools menu option.

Count=

Specifies the number of items that should appear on the Tools menu. This item is changed by adding or removing programs from the Tools Options dialog.

Title#=
Path#=
WorkingDir#=
Params#=

These entries appear once each for every item on the Tools menu. Each item name is immediately followed by a number indicating its position in the Tools menu, zero being the first.

Title#	Specifies the text that actually appears on the Tools menu.
Path#=	Specifies the full path to the program that the menu option will execute.

WorkingDir# Determines the current directory when the program starts.
Params# Specifies the parameters to pass to the program at startup.

Section: [Closed Files] - The Closed Files section lists the full path name of the last three closed project files. The files are listed in the order of most recently used first. Each entry takes the form

File_#=<projectname>.DPR,col1,row1,col2,row2

where # is either 0, 1 or 2. Col1 identifies the first visible column in the code editor, row1 the first visible row. Col2 is the cursor column, row2 the cursor row.

Section: [VBX] - The VBX section contains various settings that are available when installing a VBX into the Delphi Component Library.

VBXDir=

Contains the last location from which a VBX was installed. This value is saved automatically by Delphi upon installing a VBX.

UnitDir=

Specifies the last location in which Delphi placed a source unit for use with the previously installed VBX. This value is saved automatically by Delphi upon installing a VBX.

PalettePage=BVSP

This entry retains the last specified name of the component palette page onto which Delphi placed the most recently installed VBX. This value is saved automatically by Delphi upon installing a VBX.

Section: [Version Control]

VCSManager=

This item specifies the fully qualified path of a Version Control manager DLL. Delphi Client/Server, which includes team support, supplies a Version Control manager by the name STDVCS.DLL, located in the \BIN directory. Example:

VCSManager=d:\delphi\bin\stdvcs.dll

Section: [Resource Expert] - The Resource Expert section appears only if the Delphi Resource Expert is installed. This section has but one entry.

RCIncludePath=

Specifies the list of directories (separated by semicolons) that the expert should search to find any include files needed for resource file conversion. Example:

RCIncludePath=D:\DELPHI\WORK;D:\RESOURCE\INCLUDE

Section: [History_##] - A number of history sections, each with a unique number following the underscore, reside in DELPHI.INI. Each history section corresponds directly to a particular combo box in a Delphi dialog. Each section contains at least one entry; the Count entry, indicating the number of history items in the section. Each actual history item is named by an H, followed by its order in the history list, H0 being first. The table below indicates to which combo box the particular section belongs. Only those histories saved by Delphi are listed.

Section	Combo box location
[History_0]	'Text to find', Find Text or Replace Text dialog
[History_1]	'Replace with', Replace Text dialog
[History_2]	'Output directory', Directory/conditionals page of Project Options dialog
[History_3]	'Search path', Directory/conditionals page of Project Options dialog
[History_7]	'Conditionals', Directory/conditionals page of Project Options dialog
[History_8]	'Undo Limit', Editor options page of Environment Options dialog
[History_9]	'Right margin', Editor display page of Environment Options dialog
[History_10]	'Tab stops', Editor options page of Environment Options dialog
[History_11]	'Syntax extensions', Editor options page of Environment Options dialog
[History_12]	'Enter new line number', Go to Line Number dialog
[History_18]	'Block indent', Editor options page of Environment Options dialog
[History_20]	'File name', Open Project dialog
[History_23]	'File name', Install VBX file dialog
[History_25]	'File name', Unit file name dialog (under Install VBX)

[History_33] 'Collapse nodes', Browser page of Environment Options dialog
[History_34] 'Library path', Library page of Environment Options dialog
[History_35] 'File name', Open Library dialog
[History_36] 'File name', Save Project1 As dialog

BDE: Writing Buffer to Disk

NUMBER : 2953
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : BDE: Writing Buffer to Disk

Product: Delphi and the Borland Database Engine
Number: ?????
Versions: 1.x, 2.x
OS: WINDOWS 3.x, WINDOWS 95, WINDOWS NT
DATE: December 7, 1995
TITLE: Using DbiUseIdleTime and DbiSaveChanges.

General:
=======

Changes made to a table are not written directly to disk until the table is closed. A power failure or system crash can result in a loss of data, and an inconvenience. To avoid this loss of data, two direct Database Engine calls can be made, both of which have the similar effects. These functions are DbiUseIdleTime and DbiSaveChanges.

DbiSaveChanges(hDBICur):
========================

DbiSaveChanges saves to disk all the updates that are in the buffer of the table associated with the cursor (hDBICur). It can be called at any point. For example, one may want to make save changes to disk every time a record is updated (add dbiProcs to uses clause):

```
procedure TForm1.Table1AfterPost(DataSet: TDataSet);
begin
  DbiSaveChanges(Table1.handle);
end;
```

This way, one does not have to worry about losing data if a power failure or system crash occurs after a record update.

DbiSaveChanges can also be used to make a temporary table (created by DbiCreateTempTable) permanent.

This function does NOT apply to SQL tables.

DbiUseIdleTime:
==============

DbiUseIdleTime can be called when the "Windows Message Queue" is empty. It allows the Database Engine to save "dirty buffers" to disk. In other words, it does what DbiSaveChanges does, but

performs the operation on ALL the tables that have been updated. This operation however, will not necessarily occur after every record update, because it can only be executed when there is an idle period.

In Delphi, it can be used in this fashion (add dbiProcs to uses clause):

```
procedure TForm1.FormCreate(Sender: TObject);
begin
      Application.onIdle := UseIdle;
end;

procedure Tform1.UseIdle(Sender: TObject; var Done: Boolean);
begin
       DbiUseIdleTime;
end;
```

USAGE NOTES:
==========

Using both DbiUseIdleTime and DbiSaveChanges (after every record modification) is redundant and will result in unnecessary function calls. If the application is one that perfroms a great deal of record insertions or modifications in a small period of time, it is recommended that the client either call DbiUseIdleTime during an idle period, or call DbiSaveChanges after a group of updates.

If not very many updates are being performed on the table, the client may choose to call DbiSaveChanges after every post or set up a timer and call DbiUseIdleTime when a timer even is generated.

Creating and Using Parameterized Queries

NUMBER : 2954
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : Creating and Using Parameterized Queries

Q: How do I pass a variable to a query?

A: First, you must write a query that uses a variable.

```
Select Test."FName", Test."Salary Of Employee"
From Test
Where Test."Salary of Employee" > :val
```

Note: If you just write the field name as "Salary of Employee" you will get a Capability Not Supported error. It must be Test."Salary of Employee".

In this can the variable name is "val", but it can be whatever you want (of course). Then, you go to the TQuery's params property and set the "val" parameter to whatever the appropriate type is. In our example here we will call it an integer.

Next, you write the code that sets the parameter's value. We will be setting the value from a TEdit box.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Query1 do
  begin
    Close;
    ParamByName('val').AsInteger := StrToInt(Edit1.Text);
    Open;
  end;
end;
```

Note: you may want to place this code in a try..except block as a safety precaution.

If you want to use a LIKE in your query, you can do it this way:

Note: This next section uses the customer table from the \delphi\demos\data directory. It can also be referenced by using the DBDEMOS alias.

SQL code within the TQuery.SQL property:

```
  SELECT * FROM CUSTOMER
```

```
  WHERE Company LIKE :CompanyName
```

Delphi code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Query1 do
  begin
    Close;
    ParamByName('CompanyName').AsString := Edit1.Text + '%';
    Open;
  end;
end;
```

An alternate way of referencing a parameter
(other then ParamByName) is params[TheParameterNumber].

The way that this line:

```
    ParamByName('CompanyName').AsString := Edit1.Text + '%';
```

can be alternately written is:

```
    Params[0].AsString := Edit1.Text + '%';
```

The trick to the wildcard is in the concatenating of the
percentage sign at the end of the parameter.

Changing the NET DIR Programmatically

NUMBER : 2919
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Changing the NET DIR Programmatically

General:
The "NET DIR" parameter for the Paradox driver is usually indicated in the BDE Configuration Utility on the drivers tab. The NET DIR indicates where the PDOXUSRS.NET file is located. If the NET DIR is pointing to a location not containing a PDOXUSRS.NET file, one will be created there.

Changing the NET DIR programmatically:
It is possible to change where the NET DIR is pointing programmatically by using the DbiSetProp() function, but first, a handle to the BDE session must be acquired. The session handle can be acquired through DbiGetCurrSession(). Here is how one could change the NET DIR:

```
void main(void)
{
	hDBISes hSes;

	DbiInit(NULL);
	DbiGetCurrSession(&hSes);
	DbiSetProp(hSes, sesNETFILE, (UINT32)"c:\\temp");
	Dbi...
	...
}
```

Notes:
The sesNETFILE constant is an Object Property and a clearer definition can be found by searching for "Object Property" in the BDE On-Line help. Another way to change the default NET DIR is to pass DbiInit() a DBIEnv structure that specifies a different .cfg (configuration) file other than the default IDAPI.CFG (calling DbiInit(NULL) will use IDAPI.CFG). This method is not reccommended because only one configuration file can be used for all BDE applications.

How to Populate a TDBComboBox Or TDBListBox

NUMBER : 2956
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : How to Populate a TDBComboBox Or TDBListBox

Filling dblistboxs and dbcomboboxs

Most of Delphi's data aware components will populate themselves after they are wired up to a open dataset. However DbListboxs and DbComboboxs do not display this characteristic. These two components are not for displaying your datasets, but filling them. Use of these components is straight forward. When you update your table, the value of the DbListbox or DbCombobox will be posted in the appropriate field.

Filling the DbCombobox or DbListbox the same as filling normal comboboxs or listboxes. The lines of text in a listbox or combobox are really a tstring list. The "Items" property of the given component holds this list. Use the "Add" method for adding items to a tstring. If you want to use data types other than strings they must be converted at run time. If your list has a blank line at the end, consider setting the "IntegralHeight" property to True.

Filling a DbListbox with 4 lines programmatically might look similar to this:

```
DbListbox1.items.add('line one');
DbListbox1.items.add('line two');
DbListbox1.items.add('line three');
DbListbox1.items.add('line four');
```

Filling a DbListbox at design time requires using the object inspector. By double clicking on the components "Items" property, you can bring up the "String List Editor" and input the desired rows.

Unfortunately, if a combobox is filled this way, there is not default value. Setting a DbComboboxs "text" property will achieve this result. (the "text" property is not available in the object inspector, so it must be set programmatically). Setting the default value to the first value in the DbCombobox's list looks like this:

```
DbCombobox1.text := DbCombobox1.items[0];
```

Often it is useful to fill a DBListBox from a dataset. This can be done using loop:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with table2 do begin
    open;
    while not EOF do
    begin
      DBlistbox1.items.add(FieldByName('name').AsString);
      next;
    end;
  end;
end;
```

New Language Features in Delphi 2.0

NUMBER : 2957
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : May 21, 1996

TITLE : New Language Features in Delphi 2.0

New Language Features in Delphi 2.0 - 32

Delphi 2.0 defines several new data types that exploit the features available by Windows 95. Delphi 2.0 has also changed a few data types to take advantage of the 32-bit environment.

New data types include:

- Character type
- String type
- Variant type
- Currency type

Changed data types:

- Integer
- Cardinal

Additional Syntax:

- Unit finalization section

New Data Types

Character Type

Delphi 2.0 introduces new wide character types to support Unicode. Delphi 1.0 treated characters as 8-bit values of type Char.

These are the standard types that represent characters in Delphi 2.0.

ANSIChar - A standard 8-bit ANSI character, equivalent to the Char type in previous versions of Delphi.

WideChar - A 16-bit character, representing a Unicode character. If the high-order byte is zero, the low-order byte contains an ANSI character.

Char - By default, Char is equivalent to ANSIChar. Char works in the same way as the implementation-dependent Integer type, which is

equivalent to SmallInt in 16-bit versions of Delphi and to LongInt in 32-bit versions of Delphi. In Delphi 2.0, Char defaults to an 8-bit value.

Character-pointer types:

Pointer type	Character type
PANSIChar	ANSIChar
PWideChar	WideChar
PChar	Char

The semantics of all the character-pointer types are identical. The only thing that varies is the size of the character pointed to.

String Type

Delphi 2.0 supports strings of nearly unlimited length in addition to the 255-character counted strings previously supported. A new compiler directive, $H, controls whether the reserved word "string" represents a short string or the new, long string. The default state of $H, is $H+, using long strings by default. All Delphi 2.0 components use the new long string type.

These are the new string types.

ShortString - A counted string with a maximum length of 255 characters. Equivalent to String in Delphi 1.0. Each element is of type ANSIChar.

AnsiString - A new-style string of variable length, also called a "long string." Each element is of type ANSIChar.

string - Either a short string or an ANSI string, depending on the value of the $H compiler directive.

Here are the compatibility issues.

Although most string code works interchangeably between short strings and long strings, there are certain short-string operations that either won't work on long strings at all or which operate more efficiently when done a different way. The following table summarizes these changes.

Short String operation	Long string equivalent	Explanation
PString type	string	All long strings are dynamically allocated, so PString is redundant and requires more bookkeeping.
S[0] := L	SetLength(S,L) SetString(S,P,L)	Because long strings are dynamically allocated, you

		must call the SetLength procedure to allocate the appropriate amount of memory.
StrPCopy (Buffer, S)	StrPCopy(Buffer, PChar(S))	You can typecast long strings into null-terminated strings. The address of the long string is the address of its first character, and the long string is followed by a null.
S := StrPas(P)	S := P	Long strings can automatically copy from null-terminated strings.

Long strings cannot be passed to OpenString-type parameters or var short-string parameters.

Variant Type

Delphi 2.0 introduces variant types to give you the flexibility to dynamically change the type of a variable. This is useful when implementing OLE automation or certain kinds of database operations where the parameter types on the server are unknown to your Delphi-built client application.

A variant type is a 16-byte structure that has type information embedded in it along with its value, which can represent a string, integer, or floating-point value. The compiler recognizes the standard type identifier Variant as the declaration of a variant.

In cases where the type of a variant is incompatible with the type needed to complete an operation, the variant will automatically promote its value to a compatible value, if possible. For instance, if a variant contains an integer and you assign it to a string, the variant converts its value into the string representing the integer number, which is then assigned to the string.

You can also assign a variant expression to a variable of a standard type or pass the variant as a parameter to a routine that expects a standard type as a parameter. Delphi coerces the variant value into a compatible type if necessary, and raises an exception if it cannot create a compatible value.

Currency Type

Delphi 2.0 defines a new type called Currency, which is a floating-point type specifically designed to handle large values with great precision. Currency is assignment-compatible with all other floating-point types (and variant types), but is actually stored in a 64-bit integer value much like the Comp type.

Currency-type values have a four-decimal-place precision. That is, the floating-point value is stored in the integer format

with the four least significant digits implicitly representing four decimal places.

Changed Data Types

The implementation-dependent types Integer and Cardinal are 32-bit values in Delphi 2.0, where they were 16-bit values in Delphi 1.0. To explicitly declare 16-bit integer data types, use the SmallInt and Word types.

Additional Syntax

You can include an optional finalization section in a unit. Finalization is the counterpart of initialization, and takes place when the application shuts down. You can think of the finalization section as "exit code" for a unit. The finalization section corresponds to calls to ExitProc and AddExitProc in Delphi 1.0.

The finalization begins with the reserved word finalization. The finalization section must appear after the initialization section, but before the final end. statement.

Once execution enters an initialization section of a unit, the corresponding finalization section is guaranteed to execute when the application shuts down. Finalization sections must handle partially-initialized data properly, just as class destructors must.

Finalization sections execute in the opposite order that units were initialized. For example, if your application initializes units A, B, and C, in that order, it will finalize them in the order C, B, and A.

The outline for a unit therefore looks like this:

```
unit UnitName;
interface
{ uses clause; optional }
...
implementation
{ uses clause; optional }
...
initialization { optional }
...
finalization { optional }
...
end.
```

Preventing a Form from Resizing

NUMBER : 2958
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 4, 1996

TITLE : Preventing a Form from Resizing

In some cases, developers would want to create a regular window (Form) in Delphi that contains some of the characteristics of a dialog box. For example, they do not want to allow their users to resize the form at runtime due to user interface design issues. Other than creating the whole form as a dialog box, there is not a property or a method to handle this in a regular window in Delphi. But due to the solid connection between Delphi and the API layer, developers can accomplish this easily.

The following example demonstrates a way of handling the Windows message "WM_GetMinMaxInfo" which allows the developer to restrict the size of windows (forms) at runtime to a specific value. In this case, it will be used to disable the functionality of sizing the window (form) at runtime.

Consider the following unit:

```
unit getminmax;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
    procedure WMGetMinMaxInfo(var Msg: TWMGetMinMaxInfo);
                message WM_GETMINMAXINFO;
    procedure WMInitMenuPopup(var Msg: TWMInitMenuPopup);
                message WM_INITMENUPOPUP;
    procedure WMNCHitTest(var Msg: TWMNCHitTest);
                message WM_NCHitTest;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
```

```
procedure TForm1.WMGetMinMaxInfo(var Msg: TWMGetMinMaxInfo);
begin
      inherited;
      with Msg.MinMaxInfo^ do
      begin
            ptMinTrackSize.x:= form1.width;
            ptMaxTrackSize.x:= form1.width;
            ptMinTrackSize.y:= form1.height;
            ptMaxTrackSize.y:= form1.height;
      end;
end;

procedure TForm1.WMInitMenuPopup(var Msg: TWMInitMenuPopup);
begin
       inherited;
       if Msg.SystemMenu then
          EnableMenuItem(Msg.MenuPopup, SC_SIZE, MF_BYCOMMAND or MF_GRAYED)
end;

procedure TForm1.WMNCHitTest(var Msg: TWMNCHitTest);
begin
       inherited;
       with Msg do
            if Result in [HTLEFT, HTRIGHT, HTBOTTOM, HTBOTTOMRIGHT,
                          HTBOTTOMLEFT, HTTOP, HTTOPRIGHT, HTTOPLEFT] then
              Result:= HTNOWHERE
end;
end.   { End of Unit}
```

A message handler for the windows message "WM_GetMinMaxInfo" in the code above was used to set the minimum and maximum TrackSize of the window to equal the width and height of the form at design time. That was actually enough to disable the resizing of the window (form), but the example went on to handle another couple of messages just to make the application look professional. The first message was the "WMInitMenuPopup" and that was to gray out the size option from the System Menu so that the application does not give the impression that this functionality is available.
The second message was the "WMNCHitTest" and that was used to disable the change of the cursor icon whenever the mouse goes over one of the borders of the window (form) for the same reason which is not to give the impression that the resizing functionality is available.

SQL: Sorting on a Calculated Column

NUMBER : 2961
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : SQL: Sorting on a Calculated Column

At times, a given data schema will require that a data set will need to be ordered by the result of a calculation. In Delphi applications using SQL, this is possible, but the methodlogy varies slightly depending on the database type used.

For local SQL involving Paradox and dBASE tables, the calculated field would be given a name using the AS keyword. This allows the calculated field to be referenced for such purposes as setting a sort order with an ORDER BY clause in an SQL query. For example, using the sample table ITEMS.DB:

```
SELECT I."PARTNO", I."QTY", (I."QTY" * 100) AS TOTAL
FROM "ITEMS.DB" I
ORDER BY TOTAL
```

In this example, the calculated field is designated to be referred to as TOTAL, this column name then being available for the ORDER BY clause for this SQL statement.

The above method is not supported for InterBase. It is still possible, though, to sort on a calculated field in InterBase (IB) or the Local InterBase Server tables. Instead of using the name of the calculated field, an ordinal number representing the calculated field's position in field field list is used in the ORDER BY clause. For example, using the sample table EMPLOYEE (in the EMPLOYEE.GDB database):

```
SELECT EMP_NO, SALARY, (SALARY / 12) AS MONTHLY
FROM EMPLOYEE
ORDER BY 3 DESCENDING
```

While IB or LIBS tables require this second method and cannot use the first method described, either of the two methods can be used with local SQL. For example, using the SQL query for the Paradox table and adapting it to use the relative position of the calculated field rather than the name:

```
SELECT I."PARTNO", I."QTY", (I."QTY" * 100) AS TOTAL
FROM "ITEMS.DB" I
ORDER BY 3
```

pertains.

Working With Auto-increment Field Types

NUMBER : 2955
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : March 4, 1996

TITLE : Working With Auto-increment Field Types

In Delphi applications, the use of tables containing fields that autoincrement, or change automatically in some manner unknown to the application, can be problematic. Paradox, InterBase, Sybase and Informix tables all provide means of inserting or updating field values automatically, without intervention from the front-end application. Not every table operation is affected by this mechanism, however. So, this document will attempt to provide a guideline for dealing with issues relating to the use of such field types in Paradox 5.0, Informix 5.x, MS/Sybase SQL Server 4.x, InterBase 4.0 and Local InterBase tables.

For each table type, a different mechanism provides this behind-the-scenes behavior. Paradox tables support an Autoincrement field type. When new records are added to such tables, the Borland Database Engine determines the highest current value in that column, adds one, and updates the new row with the new value.

For Informix tables, this behavior is provided by an Informix-specific field type called Serial. Serial columns differ from Paradox Autoincrement fields in that their values may be changed, while Autoincrement columns are read-only.

InterBase and MS/Sybase SQL Server tables do not support a special type for this kind of behavior, but may employ triggers to accomplish the same task. Triggers are specialized procedures that reside on the database server and automatically execute in response to events such as table inserts, updates and deletes. The use of tables with associated triggers can be particularly problematic, since triggers are capable of doing much more than just incrementing column values.

The three areas that are affected by these field types are simple inserts, batchmoves, and table linking.

Handling Update and/or Append BatchMoves

Paradox Tables

Since the Autoincrement field type is a read-only type, attempting to perform a batchmove operation with such a column in the destination table may cause an error. To circumvent this, the TBatchMove components Mappings property must be set to match

source table fields to the target destination fields excluding the destination table's Autoincrement field.

Informix Tables

Batch moving rows to Informix tables with Serial columns will not cause an error in and of itself. However, caution should be used since Serial columns are updateable and are often used as primary keys.

InterBase Tables
MS/Sybase SQL Server Tables

Triggers on InterBase and SQL Server tables may catch any improper changes made to the table, but this depends strictly upon the checks placed in the trigger. Here again, caution should be used since trigger-updated columns are often used as primary keys.

Linking Tables via MasterSource & MasterFields

Paradox Tables
Informix Tables

If the MasterFields and MasterSource properties are used to create linked tables in a master-detail relationship and one of the fields in the detail table is an Autoincrement or Serial field, then the matching field in the master table must be a Long Integer field or a Serial field. If the master table is not a Paradox table then the master table's key field may be any integer type it supports.

InterBase Tables
MS/Sybase SQL Server Tables

Linking with these tables types presents no particular problems relating to trigger-modified fields. The only necessity is matching the appropriate column type between the two tables.

Simple Inserts/Updates

Paradox Tables

Since Paradox Autoincrement fields are read-only, they are not typically targeted for update when inserting new records. Therefore, the Required property for field components based on Autoincrement fields should always be set to False. This can be accomplished from within Delphi, using the Fields Editor to define field components at design time by double clicking on the TQuery or TTable component or at runtime with a statement similar to the following.

```
Table1.Fields[0].Required := False;
```

or

```
Table1.FieldByName('Fieldname').Required := False;
```

Informix Tables

Although Informix Serial fields are updateable, if their autoincrement feature is to be used, then the Required property of field components based on them should be set to False. Do this in the same manner described for Paradox Tables.

InterBase Tables
MS/Sybase SQL Server Tables

Handling inserts on these trigger-modified table types requires a number of steps for smooth operation. These additional steps are particularly necessary if inserts are accomplished via standard data-aware controls, such as DBEdits and DBMemos.

Inserting rows on trigger-modified InterBase and SQL Server tables may often yield the error message 'Record/Key Deleted'. This error message appears despite that the table is properly updated on the server. This will occur if:

1. The trigger updates the primary key. This is not only likely when a trigger is used, but is probably the most common reason for using a trigger.

2a. Other columns in the table have bound default values. This is accomplished with the DEFAULT clause at table creation in the case of InterBase. or with the sp_bindefault stored procedure in SQL Server.

or

2b. Blob type fields are updated when a new row is inserted.

or

2b. Calculated fields are defined in an InterBase table.

The fundamental cause for this is that when the record (or identifying key) is changed at the server, the BDE no longer has means of specifically identifying the record for re-retrieval. That is, the record no longer appears as it did when it was posted, therefore the BDE assumes that the record has been deleted (or the key changed).

Firstly, the field components of trigger-modified fields must have their Required property set to False. Do this in the same

manner described for Paradox Tables.

Secondly, to avoid the spurious error, order the table by an index that does not make use of fields updated by the trigger. This will also prevent the newly entered record from disappearing immediately after insertion.

Lastly, if requirement 1 above holds but neither 2a, 2b nor 2c hold, then code similar to the following should be used for the table component's AfterPost event handler.

```
procedure TForm1.Table1AfterPost(DataSet: TDataset);
begin
   Table1.Refresh
end;
```

A Refresh of the table is necessary to re-retrieve the values changed by the server.

If criteria 2a, 2b or 2c cannot be avoided, then the table should be updated without using Delphi's data-aware controls. This can be accomplished using a TQuery component targeted at the same table. Once the query has posted the update, any table components using the same table should be Refreshed.

Delphi 2.0 for Windows 95 & Windows NT Factsheet

NUMBER : 2996
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : May 6, 1996

TITLE : Delphi 2.0 for Windows 95 & Windows NT Factsheet

Delphi 2.0
for Windows 95 and Windows NT
Factsheet

Delphi 2.0 is the only Rapid Application Development tool that combines the performance of the world's fastest optimizing 32-bit native-code compiler, the productivity of visual component-based design, and the flexibility of scalable database architecture in a robust object-oriented environment.

"The most exciting new programming tool. Delphi combines the intuitive ease of a visual programming tool with the power of a true compiler." -PC Magazine, January 1996

The fastest way to the fastest 32-bit applications

Now, whether you are a beginning programmer, professional developer, or client/server developer, there is a version of Delphi that is right for you. Introducing Delphi Desktop 2.0, Delphi Developer 2.0, and Delphi Client/Server Suite 2.0. Delphi 2.0 for Windows 95 and Windows NT combines the most intuitive visual design environment and scalable database tools with the world's fastest optimizing 32-bit native-code compiler.

The NEW! high-performance, optimizing 32-bit native-code compiler creates standalone executables that are up to 300 to 400% faster than 16-bit Delphi applications, and up to 15 to 50 times faster than applications built with p-code interpreters. Create fast, standalone, royalty-free EXEs with no runtime interpreter DLLs required.

Only Delphi 2.0 brings the power of true 32-bit development to a visual environment. The 32-bit architecture eliminates all 16-bit memory restrictions, enabling you to build strings and data structures up to two gigabytes.

Delphi 2.0 includes FREE! Delphi 1.0 for 16-bit Windows 3.1 application development, making it a complete solution no matter where your Windows-based applications reside today. A simple recompile allows rapid migration of 16-bit applications to 32 bits, preserving your existing investment in code.

Reuse: The key to productivity

The NEW! 32-bit Visual Component Library (VCL) includes more than 90 customizable, reusable components and a dozen NEW! Windows 95 components, so you can drag-and-drop your way to your first 32-bit application in no time. Everything from tree views and list boxes, to notebook tabs, grids, and multimedia controls, is right at your fingertips. You can easily modify and reuse Delphi components and OCX controls, saving time and boosting productivity.

FREE! Quick Reports components allow you to easily create, preview, and print simple embedded reports.

NEW! Visual Form Inheritance maximizes reuse, simplifies the implementation of standards, and makes code maintenance easier. Visual Form Inheritance lets you visually create new forms that inherit all properties from another form.

Create reusable DLLs easily

Build DLLs in Delphi and use them with C++, Paradox,(R) Visual Basic, and other tools. And gain tighter integration between all of your Borland(R) C++ and Delphi applications with NEW! support for OBJ files and COM objects.

Raising the bar in Rapid Application Development

The object-oriented architecture and Two-Way-Tools(TM) in Delphi maximize productivity. Everything you do visually is automatically reflected in code.

The NEW! centralized Object Repository stores forms, Data Module Objects, and business rules for reuse. Any new application can inherit, use, or copy an existing structure so you can pick the architecture that best fits your development needs.

NEW! Data Module Objects act as your application's information core, allowing you to define data sources, store relationships, and apply business rules to your database. You can separate your business rules from application code for easier maintenance.

A comprehensive suite of visual design and debugging tools accelerate development so you move seamlessly from prototype to production. The NEW! multi-error compile with hints and warnings helps you resolve errors faster. Programmers can jump to and fix multiple problems in a single compile. For quick and easy debugging of even your largest applications, the new compiler even catches logic errors like empty loops, uninitialized pointers, and unused variables.

Unleash the power of Windows 95 and Windows NT

Delphi 2.0 provides complete access to Windows 95 and NT APIs including multi-threading, MAPI, and Unicode, so you can leverage the power of the 32-bit operating system.

NEW! OCX support lets you drag-and-drop OCX controls directly into your applications from the Delphi Component Palette. Delphi also lets you subclass OCXs for easy customization. You can also integrate OLE-enabled applications like Visual Basic, Excel, Borland C++, and Paradox, with NEW! OLE Automation controller and server support.

Drag-and-drop database development

Create high-performance, scalable database applications with integrated database support. NEW! data-aware controls let you manipulate live data at design time. Simply select the fields, and Delphi instantly builds the connections.

Let Delphi 2.0 do the database work for you with NEW! Fast Filters and Smart Lookups. Fast Filters make viewing a specific subset of your data as fast and easy as changing a property. Smart Lookups automatically provide a dropdown box to seamlessly display data from multiple tables.

The NEW! Database Explorer allows you to visually browse and manage tables and aliases without leaving the Delphi IDE.

NEW! 32-bit Borland(R) Database Engine (BDE) includes a new query engine and takes advantage of 32-bit flat address space for increased performance. It provides direct access to Visual dBASE(R) and Paradox tables, and forms a solid foundation for all your database applications.

Delphi Developer: The choice for professional programmers

If you are building sophisticated, multiuser LAN or corporate database applications, you need Delphi Developer 2.0. It provides all the power of Delphi Desktop 2.0, plus a comprehensive set of professional programming tools.

Powerful programming tools

The VCL Source Code lets you easily modify native Delphi components, and build your own custom components. Also included are sample OCXs for graphing, charting, spreadsheet, and spell-checking.

Easily add your favorite custom development tools directly to the Delphi environment with the NEW! Open Tools API. Interact with CASE tools or link to your favorite editor--whatever you need to be your most productive.

Build sophisticated, professional reports using live data at

design time with the NEW! 32-bit ReportSmith.(R)

When you're ready to distribute your applications, the integrated install utility, InstallShield(R) Express, instantly builds a professional installation program for even your most complex database applications.

Advanced database power

Quickly establish and maintain data integrity with the NEW! scalable Data Dictionary. It stores customized information about the data in your tables, extended field attributes like min, max, default values, display preferences, and edit masks.

Easily build detailed views of your data with the NEW! Multi-Object Grid. It provides codeless support for quick searching and automatic lookups, and lets you decide whether data appears in rows or individual panels. Panel views support multiple dropdowns, check boxes, and edit fields.

Leverage the power of the NEW! 32-bit BDE with complete access to the BDE API and help files. You also get complete ODBC connectivity.

Complete scalability

The scalable NEW! 32-bit Local InterBase(R) Server is included for economical off-line development and testing, and for building standalone ANSI SQL-92 compliant applications.

Delphi Client/Server Suite: Gain the competitive advantage

Delphi Client/Server Suite 2.0 gives you a complete suite of tools to build scalable, high-performance client and server applications for Windows 95, Windows NT, and Windows 3.1.

Deploy royalty-free applications across corporate database servers using the NEW! 32-bit SQL Links high-performance native drivers for InterBase, Oracle, Sybase, and MS SQL Server.

Visually browse and modify server-specific meta data including stored procedures, triggers, event alerters, and domains with the NEW! SQL Database Explorer. Gain optimum performance by testing, debugging, and tuning your SQL applications with the NEW! SQL Monitor. Build and test sophisticated multiuser client/server applications with NEW! InterBase NT database server (2-user license). Use the NEW! Visual Query Builder to visually create complex database queries and automatically generate bug-free ANSI SQL-92 commands.

NEW! Cached Updates increase server responsiveness by reducing the amount of network traffic between client and server. Scale networked applications and move data between formats and host

platforms with the NEW! Data Pump Expert. Manage complex team projects with sophisticated source code check-in, check-out, and version control using the NEW! integrated PVCS Version Manager.

Delphi Client/Server Suite 2.0 shortens every stage of the development cycle, allowing you to respond to changing business needs, and gain the competitive advantage.

Delphi 2.0 Specifications and system requirements

Optimizing Native-Code Compiler
* Compiles at more than 350,000 lines per minute
* Create fast standalone EXEs with no runtime interpreter Dynamic Link Library (DLL)
* Applications run up to 15 to 50 times faster than interpreted p-code
* Create DLLs that work with C++, Visual dBASE, Paradox, Visual Basic, PowerBuilder, and others
* Full access to Windows 95 and NT APIs
* Optimized case statements, sets, 32-bit math operations, string and file routines, and more
* Smart Linker removes unused objects and code
* Automatic form linking defines uses clauses
* Advanced math library for extended statistical and financial analysis

Visual Component Library (VCL)
* More than 90 reusable components
* Support for the latest Microsoft systems technologies including OCX, OLE Automation controller and server, MAPI, and ODBC
* Standard components for menus, buttons, masked edit fields, panels, graphics, notebook tabs, grids, outlines, list boxes, and more
* "Live" design-time data access
* VCL Source Code available separately

Object Pascal Language
* Structured, object-oriented language
* Supports inheritance, polymorphism, and encapsulation
* Control over privacy with Public, Private, Protected, and Published reserved words
* Create components with properties and events
* Use inheritance to customize any object
* Runtime type information and object persistence
* Automatic, extensible exception handling
* Support for open arrays, user-defined types, objects, and pointers
* Support for delegation and class references

Borland Database Engine
* High-performance engine with native drivers for Visual dBASE, Paradox, and Local InterBase Server
* Fully scalable support for migrating applications from desktop to client/server

* Royalty-free deployment of database engine
* ODBC support for Access, Btrieve, Excel, DB2, AS/400, Ingres, HP ALBASE/SQL, and gateways like IBM DDCS/2, Micro Decisionware, and Sybase Net-gateway (available separately)
* Local InterBase is a high-performance ANSI SQL-92 conformant SQL database server

Documentation and Help

* More than 5Mb of online help
* Manuals: Getting Started, Object Pascal Language Guide, Delphi User's Guide, Component Writer's Guide, and Database Application Developer's Guide

Minimum System Requirements

* Intel 486-based PC or higher
* Microsoft Windows 95 or Windows NT
* 8Mb of extended memory or higher
* 50Mb hard disk space
* CD-ROM drive (3.5" disks available separately)

Networks Supported

* Any Microsoft Windows 95 or NT-compatible networks

The fastest way to the fastest Windows 95/NT applications	Delphi Desktop	Delphi Developer	Delphi C/S Suite
Optimizing 32-bit native code compiler	X	X	X
32-bit Borland Database Engine (BDE)	X	X	X
Create reusable DLLs and royalty-free, standalone EXEs that are up to 15 to 50 times faster than p-code interpreters	X	X	X
Full Windows 95/NT support for OLE Automation controllers, servers, and OLE Controls (OCXs)	X	X	X
Complete access to Windows 95/NT APIs for multi-threading, Unicode, long filenames, MAPI, and more	X	X	X
Complete suite of Windows 95 common controls	X	X	X
Integrated Development Environment	X	X	X
Object-oriented, fully extensible component architecture	X	X	X
Object Repository for storing and	X	X	X

reusing forms, Data Modules, and DLLs			
Visual Form Inheritance to reduce coding and simplify maintenance	X	X	X
Visual Component Library (VCL) with more than 90 reusable components	X	X	X
Data-aware components for drag-and-drop database applications	X	X	X
Data Module Objects to separate business rules from application code	X	X	X
Database Explorer to visually browse and modify tables, aliases, and indexes	X	X	X
FREE! Quick Reports to easily create, preview, and print embedded reports	X	X	X
FREE! 16-bit Delphi 1.0 for Windows 3.1 Deployment	X	X	X
Complete documentation including Object Pascal Language Guide	X	X	X
VCL Source Code and manual to develop or customize components		X	X
Scalable Data Dictionary to implement and maintain data integrity		X	X
Complete ODBC connectivity		X	X
Additional components like the Multi-Object Grid		X	X
32-bit Local InterBase Server for off-line scalable SQL development		X	X
32-bit ReportSmith for sophisticated reporting		X	X
InstallShield Express for building professional installs		X	X
WinSight(TM) 32 for monitoring Windows messaging		X	X
BDE low-level API support and Help		X	X

files		
Expanded Open Tools API to integrate your favorite tools	X	X
Team development interface (requires InterSolv PVCS)	X	X
Versatile sample OCXs for charting, graphics, spreadsheet, and spell-checking	X	X
32-bit SQL Links native drivers, with unlimited deployment license for Oracle, Sybase, Informix, MS SQL Server, and InterBase		X
InterBase NT (2-user license) for scalable multiuser SQL development		X
SQL Database Explorer to visually manage server-specific meta data		X
SQL Monitor to test, debug, and tune SQL applications		X
Integrated InterSolv PVCS Version Manager for team development		X
Visual Query Builder to generate bug-free SQL code		X
Cached Updates to speed up server response time		X
Data Pump Expert for rapid upsizing and application scaling		X

SQL: Summarizing a Calculated Column

NUMBER : 2963
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : SQL: Summarizing a Calculated Column

Occasionally in a Delphi application that uses SQL to access data, it becomes necessary to summarize calculated data. That is, to create a calculated column and apply the SUM function to it.

When performing this operation against SQL tables (such as those for the Local InterBase Server), it is a simple matter of enclosing the calculation within the SUM function. For example, using the sample table EMPLOYEE (in the EMPLOYEE.GDB database):

```
SELECT SUM(SALARY / 12)
FROM EMPLOYEE
```

This same methodology can also be used when the returned data set is to be grouped by the value in another column with a GROUP BY clause:

```
SELECT EMP_NO, SUM(SALARY / 12)
FROM EMPLOYEE
GROUP BY EMP_NO
ORDER BY EMP_NO
```

While SQL databases support the summarization of calculated columns, local SQL will not. Other means would be needed to obtain the results, such as copying the results of a query with a calculated column to a temporary table (as with a TBatchMove component) and then using a TQuery component to summarize the data in the temporary table.

Managing Data Segment Size

NUMBER : 2964
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : March 4, 1996

TITLE : Managing Data Segment Size

Managing Data Segment Size

The "Data Segment too large" error occurs in Delphi 16 bit applications when the size of static data, stack and local heap are larger than the 64k application limit imposed by Windows. This document discusses how to identify and measure portions of your code that consume memory in the data segment, and how to manage this limited resource.

What does the data segment consist of?

Task header:	16 bytes of various Windows system information
Static data:	Contains global variables and typed constants.
Stack:	Stores local variables allocated by procedures and functions. The default stack size is 16k and can be modified in the Options|Project|Linker page.
Local heap:	Used by Windows for temporary storage and defaults to 8k. Do not set the local heap to 0. Windows may expand this area of memory if necessary.

How do I measure total data segment size?

To get the size of a 16 bit Delphi application static data, stack and local heap for a project, compile the project and select the Delphi menu item Compile|Information. The dialog will list the following, given a new project with one form:

Source compile:	12 lines
Code size:	128981 bytes
Data size:	3636 bytes
Stack size:	16384 bytes
Local Heap size:	8192 bytes

A Delphi application starts with the overhead of static data declared in units that provide the applications initial functionality. If the only global variable is the form name, the application will still consume at least 3,636 bytes. Adding a second form increases the datasize to only 3640 -- an increase of only the size of the global variable needed to declare a second form.

```
var
  Form2: TForm2; { 4 byte pointer }
```

The total data segment size, (the sum of static data, stack and local heap) is 28,212 bytes:

Data size:	3,636
Stack size:	16,384
Local Heap size:	8,192
	28,212

What portions of my project increase data size?

-- Variables declared within the interface and implementation sections.
-- Typed constants declared anywhere within the application. An example of a typed constant declaration:

```
const
  MyConst: integer = 100;
```

Units declared in the Uses clause and components contain code that may hvae global variables or typed constants. For instance, a TBitBtn adds 180 bytes when added to a project, where at least 132 bytes are accounted for by typed constants and global variables within the Buttons.Pas unit. Adding 10 more TBitBtns to the project does not increase the project size beyond the initial 180 byte increase.

The following sample unit includes comments describing memory usage:

```
unit Test;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls;

{ Functions used from the units may have global variables
  and typed constants that will increase the size of the
  data segment. }

type
   { Class objects are stored on the global heap, not the data
     segment}
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  public
   { MyInteger and MyString are stored
     on the global heap. }
```

```
    MyInteger: Integer;
    MyString: String;
  end;

const
{ MyConst is a typed constant and is stored in
  static data portion of the data segment.
  Minimize the number of typed constants. }
  MyConst: integer = 23;

var
{ Form1 is a global variable - stored in the static
  data portion of the data segment.   You should minimize
  the number and size of global variables. Form1 is
  pointer and uses only four bytes. }
  Form1: TForm1;
{ MyGlobalString will consume 256 bytes even if the
  string is only a few characters long. }
  MyGlobalString: string;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
{ MyLocal is a local variable and is not stored
  in the data segment. }
  MyLocal: String;
begin
  MyLocal := 'Test application';
  Label1.Caption := MyLocal
end;

end.
```

What is the impact of components on the data size?

Here is a list of components from the Standard, Additional, Data Access, and Data Control(partial list) pages of the component pallette. The list shows the compile size after added a single instance of the component to a new project. In descending order by data size usage:

Component	App in bytes	Bytes Over 3,636
table	4272	636
batchmove	4272	636
storedproc	4258	622
query	4250	614
database	4036	400
datasource	3886	250

outline	3838	202
bitbtn	3816	180
stringgrid	3794	158
drawgrid	3790	154
maskedit	3762	126
memo	3750	114
report	3722	86
listbox	3704	68
edit	3694	58
tabset	3692	56
combobox	3674	38
scrollbar	3654	18
button	3652	16
checkbox	3652	16
radiobutton	3652	16
radiogroup	3652	16
panel	3650	14
label	3648	12
speedbutton	3646	10
header	3644	8
scrollbox	3644	8
notebook	3638	2
menu	3636	0
groupbox	3636	0
tabbednotebook	3636	0
image	3636	0
shape	3636	0

How do I manage data segment size?

---

1) Avoid declaring global variables or typed constants, particularly large arrays. Instead, declare a type and a pointer to that type. Then use memory management routines such as Getmem to allocate memory from the global heap.
   This reduces the resource hit on the data segment to the 4 bytes used by the pointer variable. See code sample below.
2) Be aware of the impact of components. See above.
3) If you have a large number of strings, or arrays of strings, allocate these dynamically. Note: strings default to 255 in length -- declare as a specific size where possible: (i.e. MyShortString: string[20]).
4) A TStringList object can be used to create and manipulate large numbers of strings.
5) The Pchar type, "pointer to a string", can be used to dynamically create and manipulate character strings using little data segment space. See online help under "String-handling routines (Null-terminated)".
6) Information on memory issues is available in Object Pascal Language Guide, OBJLANG.ZIP and can be downloaded from Compuserve, DELPHI forum, and the World Wide Web, www.borland.com/TechInfo/delphi/whatsnew/dwnloads.html.

Alternative to global declaration of a large structure

Here's an example that will waste 32 bytes of data segment,
followed by a second example using only 4 bytes, and
accomplishes essentially the same task.

```
1)
  { Declaring TMyStructure causes no change
    in the data segment size. }
  TMyStructure = record
    Name: String[10];
    Data: array[0..9] of Integer;
  end;

 var
   Form1: TForm1;

  { MyStructure declaration causes a 32 byte
    increase:
    Mystructure pointer =   1 byte
      Name = 11 bytes (10 chars + length byte)
      Data = 20 bytes (10 * 2 bytes per integer)
  }
  MyStructure: TMyStructure;

2)
  { Declaring TMyStructure causes no change
    in the data segment size. }
    PMyStructure = ^TMyStructure;
    TMyStructure = record
      Name: String[10];
      Data: array[0..9] of Integer;
    end;

    var
      Form1: TForm1;
      { MyDataPtr causes 4 byte increase
        for the pointer variable. }
      MyDataPtr: PMyStructure;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Here, resources are taken from the heap. }
  New(MyDataPtr);
  MyDataPtr.Name := 'Fred';
  MyDataPtr.array[0] := 560;
  Dispose(MyDataPtr);
end;
```

You can also put a variable declaration within a class:

```
type
  TMyBigArray = array[1..100] of string

  TForm1 = class(TForm)
  public
    { This declaration has no impact on data segment size. }
    MyBigArray: TMyBigArray;
  end;

var
  { This declaration increases data segment by 25,600 bytes. }
  MyOtherBigArray: TMyBigArray;
```

Validating input in TEdit components

NUMBER : 2967
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1997

TITLE : Validating input in TEdit components

Q: How can I validate input in my TEdit components?

A: Assuming you're using regular TEdit components, during OnExit, you will see irregular behavior from controls if you attempt to change focus at that time.

The solution is to post a message to your form in the TEdit's OnExit event handler. This user-defined posted message will indicate that the coast is clear to begin validating input. Since posted messages are placed at the end of the message queue, this gives Windows the opportunity to complete the focus change before you attempt to change the focus back to another control.

Attached is a unit and text representation of a DFM (form) file which demonstrates this technique.

```
{ *** BEGIN CODE FOR UNIT5.PAS *** }
unit Unit5;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Mask;

const
  { User-defined message }
  um_ValidateInput = wm_User + 100;

type
  TForm5 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Button1: TButton;
    procedure EditExit(Sender: TObject);
    procedure EditEnter(Sender: TObject);
  private
    Refocusing: TObject;
    { User-defined message handler }
    procedure ValidateInput(var M: TMessage); message um_ValidateInput;
  end;
```

```
var
  Form5: TForm5;

implementation

{$R *.DFM}

procedure TForm5.ValidateInput(var M: TMessage);
var
E : TEdit;
begin
  { The following line is my validation.   I want to make sure }
  { the first character is a lower case alpha character.   Note }
  { the typecast of lParam to a TEdit. }
  E := TEdit(M.lParam);
  if not (E.Text[1] in ['a'..'z']) then begin
    Refocusing := E;                          { Avoid a loop       }
    ShowMessage('Bad input');                   { Yell at the user }
    TEdit(E).SetFocus;                         { Set focus back    }
  end;
end;

procedure TForm5.EditExit(Sender: TObject);
begin
  { Post a message to myself which indicates it's time to   }
  { validate the input.   Pass the TEdit instance (Self) as }
  { the message lParam. }
  if Refocusing = nil then
    PostMessage(Handle, um_ValidateInput, 0, longint(Sender));
end;

procedure TForm5.EditEnter(Sender: TObject);
begin
if Refocusing = Sender then
  Refocusing := nil;
end;

end.
{ *** END CODE FOR UNIT5.PAS *** }

{ *** BEGIN CODE FOR UNIT5.DFM *** }
object Form5: TForm5
  Left = 489
  Top = 303
  Width = 318
  Height = 205
  Caption = 'Form5'
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 16
  object Edit1: TEdit
    Left = 32
    Top = 32
```

```
      Width = 121
      Height = 24
      TabOrder = 0
      Text = 'Edit1'
      OnEnter = EditEnter
      OnExit = EditExit
    end
    object Edit2: TEdit
      Left = 160
      Top = 32
      Width = 121
      Height = 24
      TabOrder = 1
      Text = 'Edit2'
      OnEnter = EditEnter
      OnExit = EditExit
    end
    object Edit3: TEdit
      Left = 32
      Top = 64
      Width = 121
      Height = 24
      TabOrder = 2
      Text = 'Edit3'
      OnEnter = EditEnter
      OnExit = EditExit
    end
    object Edit4: TEdit
      Left = 160
      Top = 64
      Width = 121
      Height = 24
      TabOrder = 3
      Text = 'Edit4'
      OnEnter = EditEnter
      OnExit = EditExit
    end
    object Button1: TButton
      Left = 112
      Top = 136
      Width = 89
      Height = 33
      Caption = 'Button1'
      TabOrder = 4
    end
  end
  { *** END CODE FOR UNIT5.DFM *** }
```

DDE: A simple example

NUMBER : 2970
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : May 21, 1996

TITLE : DDE: A simple example

Q: How can I do DDE under Delphi using API calls ?

A: Its fairly easy to accomplish, following is an example of how to have a client program talk with a server program. Both are completely written in Delphi. In total there are 2 projects, 3 forms, and 3 units. This demo uses DDE ML API methods to handle the DDE requests.

The server must be running before the client will load. This demo program shows 3 different ways data can be moved between a client and a server.

1. The Client can 'POKE' data to the server.
2. The Server can automaticaly pass data to the Client and the Client will update a graph based on the results from the Server.
3. The Server's Data changes, then the Client will make a request to the Server for the new data, then update the graph.

***** How to handle the program. *****

Following are 8 files concatenated together. Each one has a { *** BEGIN CODE FOR FILENAME.EXT *** } CODE { *** END CODE FOR FILENAME.EXT *** } take each block of code BETWEEN THE { *** } lines and place in a file of the corresponding name, then compile and have fun !!!!

```
{ *** BEGIN CODE FOR DDEMLCLI.DPR *** }
program Ddemlcli;

uses
  Forms,
  Ddemlclu in 'DDEMLCLU.PAS' {Form1};

{$R *.RES}

begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
{ ***    END CODE FOR DDEMLCLI.DPR *** }


{ *** BEGIN CODE FOR DDEMLCLU.DFM *** }
object Form1: TForm1
  Left = 197
  Top = 95
```

```
  Width = 413
  Height = 287
  HorzScrollBar.Visible = False
  VertScrollBar.Visible = False
  Caption = 'DDEML Demo, Client Application'
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  Menu = MainMenu1
  PixelsPerInch = 96
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  OnShow = FormShow
  TextHeight = 16
  object PaintBox1: TPaintBox
    Left = 0
    Top = 0
    Width = 405
    Height = 241
    Align = alClient
    Color = clWhite
    ParentColor = False
    OnPaint = PaintBox1Paint
  end
  object MainMenu1: TMainMenu
    Top = 208
    object File1: TMenuItem
      Caption = '&File'
      object exit1: TMenuItem
        Caption = 'E&xit'
        OnClick = exit1Click
      end
    end
    object DDE1: TMenuItem
      Caption = '&DDE'
      object RequestUpdate1: TMenuItem
        Caption = '&Request an Update'
        OnClick = RequestUpdate1Click
      end
      object AdviseofChanges1: TMenuItem
        Caption = '&Advise of Changes'
        OnClick = AdviseofChanges1Click
      end
      object N1: TMenuItem
        Caption = '-'
      end
      object PokeSomeData: TMenuItem
        Caption = '&Poke Some Data'
        OnClick = PokeSomeDataClick
      end
    end
  end
end
{ ***    END CODE FOR DDEMLCLU.DFM *** }
```

```
{ *** BEGIN CODE FOR DDEMLCLU.PAS *** }
{*************************************************}
{                                                 }
{    Delphi 1.0 DDEML Demonstration Program       }
{    Copyright (c) 1996 by Borland International  }
{                                                 }
{*************************************************}

{ This is a sample application demonstrating the use of the DDEML APIs in
   a client application.   It uses the DataEntry server application that
   is part of this demo in order to maintain a display of the entered data
   as a bar graph.

   You must run the server application first (in DDEMLSRV.PAS), and then
   run this client.   If the server is not running, this application will
   fail trying to connect.

   The interface to the server is defined by the list of names (Service,
   Topic, and Items) in the separate unit called DataEntry (DATAENTR.TPU).
   The server makes the Items available in cf_Text format; they are con-
   verted and stored locally as integers.
}

unit Ddemlclu;

interface
uses
   SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
   Forms, Dialogs, VBXCtrl, ExtCtrls, DDEML, Menus, StdCtrls;

const
   NumValues = 3;

type

      { Data Structure which constitutes a sample }
   TDataSample = array [1..NumValues] of Integer;
   TDataString = array [0..20] of Char;       { Size of Item as text }

      { Main Form }
   TForm1 = class(TForm)
      MainMenu1: TMainMenu;
      File1: TMenuItem;
      exit1: TMenuItem;
      DDE1: TMenuItem;
      RequestUpdate1: TMenuItem;
      AdviseofChanges1: TMenuItem;
      PokeSomeData: TMenuItem;
      N1: TMenuItem;
      PaintBox1: TPaintBox;
      procedure FormCreate(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
      procedure RequestUpdate1Click(Sender: TObject);
      procedure FormShow(Sender: TObject);
      procedure AdviseofChanges1Click(Sender: TObject);
      procedure PokeSomeDataClick(Sender: TObject);
```

```
    procedure Request(HConversation: HConv);
    procedure exit1Click(Sender: TObject);
    procedure PaintBox1Paint(Sender: TObject);

  private
    { Private declarations }
  public
    Inst: Longint;
    CallBackPtr: ^TCallback;
    ServiceHSz : HSz;
    TopicHSz     : HSz;
    ItemHSz      : array [1..NumValues] of HSz;
    ConvHdl      : HConv;

    DataSample : TDataSample;
  end;

var Form1: TForm1;

implementation

const
  DataEntryName : PChar = 'DataEntry';
  DataTopicName : PChar = 'SampledData';
  DataItemNames : array [1..NumValues] of pChar = ('DataItem1',
                                                   'DataItem2',
                                                   'DataItem3');
{$R *.DFM}

{ Local Function: CallBack Procedure for DDEML }

function CallbackProc(CallType, Fmt: Word; Conv: HConv; hsz1, hsz2: HSZ;
  Data: HDDEData; Data1, Data2: Longint): HDDEData; export;
begin
  CallbackProc := 0;      { See if proved otherwise }

  case CallType of
    xtyp_Register:
      begin
        { Nothing ... Just return 0 }
      end;
    xtyp_Unregister:
      begin
        { Nothing ... Just return 0 }
      end;
    xtyp_xAct_Complete:
      begin
        { Nothing ... Just return 0 }
      end;
    xtyp_Request, Xtyp_AdvData:
      begin
        Form1.Request(Conv);
        CallbackProc := dde_FAck;
      end;
    xtyp_Disconnect:
```

```
        begin
          ShowMessage('Disconnected!');
          Form1.Close;
        end;
  end;
end;


{ Posts a DDE request to obtain cf_Text data from the server.   Requests
  the data for all fields of the DataSample, and invalidates the window
  to cause the new data to be displayed.   Obtains the data from the
  Server synchronously, using DdeClientTransaction.
}
procedure TForm1.Request(HConversation: HConv);
var
  hDdeTemp : HDDEData;
  DataStr   : TDataString;
  Err, I    : Integer;
begin
  if HConversation <> 0 then begin
    for I := Low(ItemHSz) to High(ItemHSz) do begin
      hDdeTemp := DdeClientTransaction(nil, 0, HConversation, ItemHSz[I],
        cf_Text, xtyp_Request, 0, nil);
      if hDdeTemp <> 0 then   begin
        DdeGetData(hDdeTemp, @DataStr, SizeOf(DataStr), 0);
        Val(DataStr, DataSample[I], Err);
      end; { if }
  end; { for }
 Paintbox1.Refresh;   { Redisplay the Screen }
  end; { if }
end;


procedure TForm1.FormCreate(Sender: TObject);
var
  I : Integer;
{ Constructs an instance of the DDE Client Window.   Constructs the
  window using the inherited constructor, then initializes the instance
  data.
}
begin
  Inst          := 0;        { Must be zero for first call to DdeInitialize }
  CallBackPtr:= nil;     { MakeProcInstance is called in SetupWindow     }
  ConvHdl      := 0;
  ServiceHSz := 0;
  TopicHSz    := 0;
  for I := Low(DataSample) to High(DataSample) do begin
    ItemHSz[I]      := 0;
    DataSample[I] := 0;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
{ Destroys an instance of the Client window.   Frees the DDE string
  handles, and frees the callback proc instance if they exist.   Also
  calls DdeUninitialize to terminate the conversation.   Then calls on
```

```
   the ancestral destructor to finish the job.
}
var I : Integer;
begin
   if ServiceHSz <> 0 then
      DdeFreeStringHandle(Inst, ServiceHSz);
   if TopicHSz <> 0 then
      DdeFreeStringHandle(Inst, TopicHSz);
   for I := Low(ItemHSz) to High(ItemHSz) do
      if ItemHSz[I] <> 0 then
         DdeFreeStringHandle(Inst, ItemHSz[I]);

   if Inst <> 0 then
      DdeUninitialize(Inst);     { Ignore the return value }

   if CallBackPtr <> nil then
      FreeProcInstance(CallBackPtr);
end;

procedure TForm1.RequestUpdate1Click(Sender: TObject);
begin
{ Generate a DDE Request in response to the DDE | Request menu selection.}
   Request(ConvHdl);
end;

procedure TForm1.FormShow(Sender: TObject);
{ Completes the initialization of the DDE Server Window.    Performs those
   actions which require a valid window.    Initializes the use of the DDEML.
}
var
   I       : Integer;
   InitOK: Boolean;
begin
   CallBackPtr := MakeProcInstance(@CallBackProc, HInstance);

{ Initialize the DDE and setup the callback function. If server is not
   present, call will fail.
}
   if CallBackPtr <> nil then begin
      if DdeInitialize(Inst, TCallback(CallBackPtr), AppCmd_ClientOnly,
         0) = dmlErr_No_Error then begin
         ServiceHSz:= DdeCreateStringHandle(Inst, DataEntryName, cp_WinAnsi);
         TopicHSz   := DdeCreateStringHandle(Inst, DataTopicName, cp_WinAnsi);
         InitOK := True;
{        for I := Low(DataItemNames) to High(DataItemNames) do begin }
         for I := 1 to NumValues do begin
            ItemHSz[I]:= DdeCreateStringHandle(Inst, DataItemNames[I],
               cp_WinAnsi);
            InitOK := InitOK and (ItemHSz[I] <> 0);
         end;

         if (ServiceHSz <> 0) and (TopicHSz <> 0) and InitOK then begin
            ConvHdl := DdeConnect(Inst, ServiceHSz, TopicHSz, nil);
            if ConvHdl = 0 then begin
               ShowMessage('Can not start Conversation!');
               Close;
```

```
          end
        end
        else begin
          ShowMessage('Can not create Strings!');
          Close;
        end
      end
      else begin
        ShowMessage('Can not Initialie!');
        Close;
      end;
  end;
end;

procedure TForm1.AdviseofChanges1Click(Sender: TObject);
{ Toggles the state of the DDE Advise setting in response to the
  DDE | Advise menu selection.   When this is selected, all three
  Items are set for Advising.
}
var
  I: Integer;
  TransType: Word;
  TempResult: Longint;
begin
  with TMenuITem(Sender) do begin
    Checked := not Checked;
    if Checked then
        TransType:= (xtyp_AdvStart or xtypf_AckReq)
    else
        TransType:= xtyp_AdvStop;
  end; { with }

  for I := Low(ItemHSz) to High(ItemHSz) do
    if DdeClientTransaction(nil, 0, ConvHdl, ItemHSz[I], cf_Text,
         TransType, 1000, @TempResult) = 0 then
      ShowMessage('Can not perform Advise Transaction');

  if TransType and xtyp_AdvStart <> 0 then Request(ConvHdl);
end;

procedure TForm1.PokeSomeDataClick(Sender: TObject);
{ Generates a DDE Poke transaction in response to the DDE | Poke
  menu selection.   Requests a value from the user that will be
  poked into DataItem1 as an illustration of the Poke function.
}
var
   DataStr: pChar;
   S: String;
begin
  S := '0';
  if InputQuery('PokeData', 'Enter Value to Poke', S) then begin
        S := S + #0;
        DataStr := @S[1];
        DdeClientTransaction(DataStr, StrLen(DataStr) + 1, ConvHdl,
          ItemHSz[1], cf_Text, xtyp_Poke, 1000, nil);
        Request(ConvHdl);
```

```
          end;
end;

procedure TForm1.exit1Click(Sender: TObject);
begin
    close;
end;

procedure TForm1.PaintBox1Paint(Sender: TObject);
{ Repaints the window on request.   Plots a graph of the current sales
  volume.
}
const
    LMarg = 30;      { Left Margin of graph }
var
    I,
    Norm: Integer;
    Wd: Integer;
    Step : Integer;

    ARect: TRect;

begin
  Norm := 0;
  for I := Low(DataSample) to High(DataSample) do begin
     if abs(DataSample[I]) > Norm then
        Norm := abs(DataSample[I]);
     end; { for }

  if Norm = 0 then Norm := 1;     { Just in case we have all zeros }

  with TPaintBox(Sender).Canvas do begin
          { Paint Background }
      Brush.color   := clWhite;
      FillRect(ClipRect);

          { Draw Axis }
      MoveTo(0, ClipRect.Bottom div 2);
      LineTo(ClipRect.Right, ClipRect.Bottom div 2);

      MoveTo(LMarg, 0);
      LineTo(LMarg, ClipRect.Bottom);

          { Print Left MArgin Text }
      TextOut(0,0, IntToStr(Norm));
      TextOut(0, ClipRect.Bottom div 2, '0');
      TextOut(0, ClipRect.Bottom + Font.Height, IntToStr(-Norm));

      TextOut(0, ClipRect.Bottom div 2, '0');
      TextOut(0, ClipRect.Bottom div 2, '0');
      TextOut(0, ClipRect.Bottom div 2, '0');
          { Print X Axis Text }

          { Now draw the bars based on that Normalized value.   Compute the
            width of the bars so that all will fit in the window, and
            compute an inter-bar space that is approximately 20% of the
```

```
            width of a bar.
           }
{           SelectObject(PaintDC, CreateSolidBrush(RGB(255, 0, 0)));
           SetBkMode(PaintDC, Transparent);
}
       ARect := ClipRect;
       Wd   := (ARect.Right - LMarg) div NumValues;
       Step := Wd div 5;
       Wd   := Wd - Step;
       with ARect do begin
           Left := LMarg + (Step div 2);
           Top := ClipRect.Bottom div 2;
           end; { with }

           { Display Bars and X-Axis Text }
       For i := Low(DataSample) to High(DataSample) do begin
              with ARect do begin
                  Right := Left + Wd;
                  Bottom:= Top - Round((Top-5) * (DataSample[I] / Norm));
                  end; { with }
                  { Fill Bar }
              Brush.color   := clFuchsia;
              FillRect(ARect);
                  { Display Text - Horizontal Axis }
              Brush.color   := clWhite;
              TextOut(ARect.Left, ClipRect.Bottom div 2 - Font.Height,
                StrPas(DataItemNames[i]));
              with ARect do
                  Left := Left + Wd + Step;
           end; { for }
       end; { with }
end;
end.{ ***     END CODE FOR DDEMLCLU.PAS *** }



{ *** BEGIN CODE FOR DDEMLSVR.DPR *** }
program Ddemlsvr;

uses
  Forms,
  Ddesvru in 'DDESVRU.PAS' {Form1},
  Ddedlg in '\DELPHI\BIN\DDEDLG.PAS' {DataEntry};

{$R *.RES}

begin
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TDataEntry, DataEntry);
  Application.Run;
end.
{ ***     END CODE FOR DDEMLSVR.DPR *** }

{ *** BEGIN CODE FOR DDESVRU.DFM *** }
object Form1: TForm1
  Left = 712
```

```
Top = 98
Width = 307
Height = 162
Caption = 'DDEML Demo, Serve Application'
Color = clWhite
Font.Color = clWindowText
Font.Height = -13
Font.Name = 'System'
Font.Style = []
Menu = MainMenu1
PixelsPerInch = 96
OnCreate = FormCreate
OnDestroy = FormDestroy
OnShow = FormShow
TextHeight = 16
object Label1: TLabel
  Left = 0
  Top = 0
  Width = 99
  Height = 16
  Caption = 'Current Values:'
end
object Label2: TLabel
  Left = 16
  Top = 24
  Width = 74
  Height = 16
  Caption = 'Data Item1:'
end
object Label3: TLabel
  Left = 16
  Top = 40
  Width = 74
  Height = 16
  Caption = 'Data Item2:'
end
object Label4: TLabel
  Left = 16
  Top = 56
  Width = 74
  Height = 16
  Caption = 'Data Item3:'
end
object Label5: TLabel
  Left = 0
  Top = 88
  Width = 265
  Height = 16
  Caption = 'Select Data|Enter Data to change values.'
end
object Label6: TLabel
  Left = 96
  Top = 24
  Width = 8
  Height = 16
  Caption = '0'
```

```
  end
  object Label7: TLabel
    Left = 96
    Top = 40
    Width = 8
    Height = 16
    Caption = '0'
  end
  object Label8: TLabel
    Left = 96
    Top = 56
    Width = 8
    Height = 16
    Caption = '0'
  end
  object MainMenu1: TMainMenu
    Left = 352
    Top = 24
    object File1: TMenuItem
      Caption = '&File'
      object Exit1: TMenuItem
        Caption = '&Exit'
        OnClick = Exit1Click
      end
    end
    object Data1: TMenuItem
      Caption = '&Data'
      object EnterData1: TMenuItem
        Caption = '&Enter Data'
        OnClick = EnterData1Click
      end
      object Clear1: TMenuItem
        Caption = '&Clear'
        OnClick = Clear1Click
      end
    end
  end
end
{ ***    END CODE FOR DDESVRU.DFM *** }

{ *** BEGIN CODE FOR DDESVRU.PAS *** }
{*************************************************}
{                                                 }
{   Delphi 1.0 DDEML Demonstration Program        }
{   Copyright (c) 1996 by Borland International   }
{                                                 }
{*************************************************}

{ This sample application uses the DDEML library in the server side of a
  cooperative application.  This server is a simple data-entry application
  which allows an operator to enter three data items, which are made
  available through DDE to interested clients.

  This server makes its service available under the following names:

        Service: 'DataEntry'
```

```
        Topic   : 'SampledData'
        Items   : 'DataItem1', 'DataItem2', 'DataItem3'

   Conceivably, other topics under this service could be defined.  Things
   such as historical data, information about the sampling, and so on
   might make useful topics.

   You must run this server BEFORE running the client (DDEMLCLI.PAS), or
   the client will fail the connection.

   The interface to this server is defined by the list of names (Service,
   Topic, and Items) in the separate unit called DataEntry (DATAENTR.TPU).
   The server makes the Items available in cf_Text format; they can be
   converted and stored locally as integers by the client.
}
unit Ddesvru;

interface

uses
   SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
   Forms, Dialogs, StdCtrls, Menus,

   DDEML,       { DDE APi }
   ShellApi;

const
   NumValues = 3;
   DataItemNames : array [1..NumValues] of PChar = ('DataItem1',
                                                    'DataItem2',
                                                    'DataItem3');
type
   TDataString = array [0..20] of Char;       { Size of Item as text }
   TDataSample = array [1..NumValues] of Integer;


{type
{ Data Structure which constitutes a sample }
{   TDataSample = array [1..NumValues] of Integer;
{   TDataString = array [0..20] of Char;       { Size of Item as text }

const
   DataEntryName: PChar = 'DataEntry';
   DataTopicName: PChar = 'SampledData';

type
   TForm1 = class(TForm)
      MainMenu1: TMainMenu;
      File1: TMenuItem;
      Exit1: TMenuItem;
      Data1: TMenuItem;
      EnterData1: TMenuItem;
      Clear1: TMenuItem;
      Label1: TLabel;
      Label2: TLabel;
      Label3: TLabel;
```

```
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    procedure Exit1Click(Sender: TObject);

    function MatchTopicAndService(Topic, Service: HSz): Boolean;
    function MatchTopicAndItem(Topic, Item: HSz): Integer;
    function WildConnect(Topic, Service: HSz; ClipFmt: Word): HDDEData;
    function AcceptPoke(Item: HSz; ClipFmt: Word;
      Data: HDDEData): Boolean;
    function DataRequested(TransType: Word; ItemNum: Integer;
      ClipFmt: Word): HDDEData;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure EnterData1Click(Sender: TObject);
    procedure Clear1Click(Sender: TObject);

  private
    Inst          : Longint;
    CallBack      : TCallback;
    ServiceHSz : HSz;
    TopicHSz      : HSz;
    ItemHSz       : array [1..NumValues] of HSz;
    ConvHdl       : HConv;
    Advising      : array [1..NumValues] of Boolean;

    DataSample : TDataSample;

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
uses DDEDlg; { DataEntry Form }

{$R *.DFM}

procedure TForm1.Exit1Click(Sender: TObject);
begin
   Close;
end;
{ Initialized globals }

const
  DemoTitle     : PChar = 'DDEML Demo, Server Application';

  MaxAdvisories = 100;
  NumAdvLoops : Integer = 0;
```

```
{ Local Function: CallBack Procedure for DDEML }

{ This callback procedure responds to all transactions generated by the
  DDEML.   The target Window object is obtained from the stored global,
  and the appropriate methods within that objects are used to respond
  to the given transaction, as indicated by the CallType parameter.
}
function CallbackProc(CallType, Fmt: Word; Conv: HConv; HSz1, HSz2: HSZ;
  Data: HDDEData; Data1, Data2: Longint): HDDEData; export;
var
  ItemNum    : Integer;
begin
  CallbackProc := 0;      { See if proved otherwise }


  case CallType of

    xtyp_WildConnect:
      CallbackProc := Form1.WildConnect(HSz1, HSz2, Fmt);

    xtyp_Connect:
      if Conv = 0 then
      begin
        if Form1.MatchTopicAndService(HSz1, HSz2) then
          CallbackProc := 1;    { Connected! }
      end;
{ When a connection is confirmed, record the conversation handle as the
  window's own.
}
    xtyp_Connect_Confirm:
      Form1.ConvHdl := Conv;

{ The client has requested data, either as a direct request or
  in response to an advisory.   Return the current state of the
  data.
}
    xtyp_AdvReq, xtyp_Request:
      begin
        ItemNum := Form1.MatchTopicAndItem(HSz1, HSz2);
        if ItemNum > 0 then
          CallbackProc := Form1.DataRequested(CallType, ItemNum, Fmt);
      end;

{ Respond to Poke requests ... this demo only allows Pokes of DataItem1.
  Return dde_FAck to acknowledge the receipt, 0 otherwise.
}
    xtyp_Poke:
      begin
        if Form1.AcceptPoke(HSz2, Fmt, Data) then
          CallbackProc := dde_FAck;
      end;

{ The client has requested the start of an advisory loop.   Note
  that we assume a "hot" loop.   Set the Advising flag to indicate
  the open loop, which will be checked whenever the data is changed.
}
```

```
      xtyp_AdvStart:
        begin
          ItemNum := Form1.MatchTopicAndItem(HSz1, HSz2);
          if ItemNum > 0 then begin
            if NumAdvLoops < MaxAdvisories then begin { Arbitrary number }
              Inc(NumAdvLoops);
              Form1.Advising[ItemNum] := True;
              CallbackProc := 1;
            end;
          end;
        end;

{ The client has requested the advisory loop to terminate.
}
      xtyp_AdvStop:
        begin
          ItemNum := Form1.MatchTopicAndItem(HSz1, HSz2);
          if ItemNum > 0 then
          begin
            if NumAdvLoops > 0 then
            begin
              Dec(NumAdvLoops);
              if NumAdvLoops = 0 then
                Form1.Advising[ItemNum] := False;
              CallbackProc := 1;
            end;
          end;
        end;
    end;   { Case CallType }

end;

{ Returns True if the given Topic and Service match those supported
   by this application.   False otherwise.
}
function TForm1.MatchTopicAndService(Topic, Service: HSz): Boolean;
begin
   Result := False;
   if DdeCmpStringHandles(TopicHSz, Topic) = 0 then
      if DdeCmpStringHandles(ServiceHSz, Service) = 0 then
         Result := True;
end;

{ Determines if the given Topic and Item match one supported by this
   application.   Returns the Item Number of the supported item (in the
   range 1..NumValues) if one is found, and zero if no match.
}
function TForm1.MatchTopicAndItem(Topic, Item: HSz): Integer;
var
   I : Integer;
begin
   Result := 0;
   if DdeCmpStringHandles(TopicHSz, Topic) = 0 then
      for I := 1 to NumValues do
         if DdeCmpStringHandles(ItemHSz[I], Item) = 0 then
            Result := I;
```

```
end;

{ Responds to wildcard connect requests.   These requests are generated
  whenever a client tries to connect to a server with either service or
  topic name set to 0.   If a server detects a wild card match, it
  returns a handle to an array of THSZPair's containing the matching
  supported Service and Topic.
}
function TForm1.WildConnect(Topic, Service: HSz; ClipFmt: Word): HDDEData;
var
  TempPairs: array [0..1] of THSZPair;
  Matched   : Boolean;
begin
  TempPairs[0].hszSvc   := ServiceHSz;
  TempPairs[0].hszTopic:= TopicHSz;
  TempPairs[1].hszSvc   := 0;       { 0-terminate the list }
  TempPairs[1].hszTopic:= 0;

  Matched := False;

  if (Topic= 0) and (Service = 0) then
    Matched := True                          { Complete wildcard }
  else
    if (Topic = 0) and (DdeCmpStringHandles(Service, ServiceHSz) = 0) then
      Matched := True
    else
      if (DdeCmpStringHandles(Topic, TopicHSz) = 0) and (Service = 0) then
        Matched := True;

  if Matched then
    WildConnect := DdeCreateDataHandle(Inst, @TempPairs, SizeOf(TempPairs),
      0, 0, ClipFmt, 0)
  else
    WildConnect := 0;
end;

{ Accepts and acts upon Poke requests from the Client.   For this
  demonstration, allows only the value of DataItem1 to be changed by a
  Poke.
}
function TForm1.AcceptPoke(Item: HSz; ClipFmt: Word;
  Data: HDDEData): Boolean;
var
  DataStr    : TDataString;
  Err        : Integer;
  TempSample: Integer;
begin
  if (DdeCmpStringHandles(Item, ItemHSz[1]) = 0) and
     (ClipFmt = cf_Text) then
  begin
    DdeGetData(Data, @DataStr, SizeOf(DataStr), 0);
    Val(DataStr, TempSample, Err);

    if IntToStr(TempSample) <> Label6.Caption then begin
      Label6.Caption := IntToStr(TempSample);
      DataSample[1] := TempSample;
```

```
        if Advising[1] then
          DdePostAdvise(Inst, TopicHSz, ItemHSz[1]);
      end;
      AcceptPoke := True;
    end
    else
      AcceptPoke := False;
end;

{ Returns the data requested by the given TransType and ClipFmt values.
   This could happen either in response to either an xtyp_Request or an
   xtyp_AdvReq.   The ItemNum parameter indicates which of the supported
   items (in the range 1..NumValues) was requested (note that this method
   assumes that the caller has already established validity and ID of the
   requested item using MatchTopicAndItem).   The corresponding data from
   the DataSample instance variable is converted to text and returned.
}
function TForm1.DataRequested(TransType: Word; ItemNum: Integer;
   ClipFmt: Word): HDDEData;
var ItemStr: TDataString;     { Defined in DataEntry.TPU }

begin
   if ClipFmt = cf_Text then
   begin
      Str(DataSample[ItemNum], ItemStr);
      DataRequested := DdeCreateDataHandle(Inst, @ItemStr,
        StrLen(ItemStr) + 1, 0, ItemHSz[ItemNum], ClipFmt, 0);
   end
   else
      DataRequested := 0;
end;


{ Constructs an instance of the DDE Server Window.   Calls on the
   inherited constructor, then sets up this objects own instandce
   data.
}
procedure TForm1.FormCreate(Sender: TObject);
var I : Integer;
begin
   Inst        := 0;        { Must be zero for first call to DdeInitialize }
   @CallBack := nil;      { MakeProcInstance is called in SetupWindow      }

   for I := 1 to NumValues do begin
      DataSample[I] := 0;
      Advising[I]   := False;
      end; { for }

end;



{ Destroys an instance of the DDE Server Window.   Checks to see if the
   Callback Proc Instance had been created, and frees it if so.   Also
   calls DdeUninitialize to terminate the conversation.   Then just calls
   on the ancestral destructor to finish.
```

```
}
procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  if ServiceHSz <> 0 then
    DdeFreeStringHandle(Inst, ServiceHSz);
  if TopicHSz <> 0 then
    DdeFreeStringHandle(Inst, TopicHSz);
  for I := 1 to NumValues do
    if ItemHSz[I] <> 0 then
      DdeFreeStringHandle(Inst, ItemHSz[I]);

  if Inst <> 0 then
    DdeUninitialize(Inst);     { Ignore the return value }

  if @CallBack <> nil then
    FreeProcInstance(@CallBack);
end;

procedure TForm1.FormShow(Sender: TObject);
var
  I : Integer;
{ Completes the initialization of the DDE Server Window.   Initializes
  the use of the DDEML by registering the services provided by this
  application.   Recall that the actual names used to register are
  defined in a separate unit (DataEntry), so that they can be used
  by the client as well.
}
begin
  @CallBack:= MakeProcInstance(@CallBackProc, HInstance);

  if DdeInitialize(Inst, CallBack, 0, 0) = dmlErr_No_Error then begin
    ServiceHSz:= DdeCreateStringHandle(Inst, DataEntryName, cp_WinAnsi);
    TopicHSz   := DdeCreateStringHandle(Inst, DataTopicName, cp_WinAnsi);
    for I := 1 to NumValues do
      ItemHSz[I] := DdeCreateStringHandle(Inst, DataItemNames[I],
        cp_WinAnsi);

    if DdeNameService(Inst, ServiceHSz, 0, dns_Register) = 0 then
      ShowMessage('Registration failed.');
  end;
end;

procedure TForm1.EnterData1Click(Sender: TObject);
{ Activates the data-entry dialog, and updates the stored
  data when complete.
}
var
  I: Integer;

begin
  if DataEntry.ShowModal = mrOk then begin
    with DataEntry do begin
      Label6.Caption := S1;
      Label7.Caption := S2;
```

```
        Label8.Caption := S3;
        DataSample[1] := StrToInt(S1);
        DataSample[2] := StrToInt(S2);
        DataSample[3] := StrToInt(S3);
        end; { with }

      for I := 1 to NumValues do
        if Advising[I] then
          DdePostAdvise(Inst, TopicHSz, ItemHSz[I]);
      end; { if }
end;

procedure TForm1.Clear1Click(Sender: TObject);
{ Clears the current data.
}
var
  I: Integer;

begin
  for I := 1 to NumValues do begin
    DataSample[I] := 0;
    if Advising[I] then
      DdePostAdvise(Inst, TopicHSz, ItemHSz[I]);
  end;

  Label6.Caption := '0';
  Label7.Caption := '0';
  Label8.Caption := '0';
end;



end.
{ ***     END CODE FOR DDESVRU.PAS *** }

{ *** BEGIN CODE FOR DDEDLG.DFM *** }
object DataEntry: TDataEntry
  Left = 488
  Top = 132
  ActiveControl = OKBtn
  BorderStyle = bsDialog
  Caption = 'Data Entry'
  ClientHeight = 264
  ClientWidth = 199
  Font.Color = clBlack
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  PixelsPerInch = 96
  Position = poScreenCenter
  OnShow = FormShow
  TextHeight = 13
  object Bevel1: TBevel
    Left = 8
    Top = 8
    Width = 177
```

```
    Height = 201
    Shape = bsFrame
    IsControl = True
  end
  object OKBtn: TBitBtn
    Left = 16
    Top = 216
    Width = 69
    Height = 39
    Caption = '&OK'
    ModalResult = 1
    TabOrder = 3
    OnClick = OKBtnClick
    Glyph.Data = {
      BE060000424DBE06000000000000360400002800000024000000120000000100
      0800000000008802000000000000000000000000000000000000000000000000
      80000080000000808000800000008000800080800000C0C0C000C0DCC000F0CA
      A600000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000000000
      000000000000000000000000000000000000F0FBFF00A4A0A000808080000000
      FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00030303030303
      0303030303030303030303030303030303030303030303030303030303030303
      03030303030303030303030303030303030303030303FF030303030303030303
      0303030303030304040303030303030303030303030303030303F8F8FF03030303
      03030303030303030303030304020204030303030303030303030303F80303F8
      FF030303030303030303030303040202020202040303030303030303030303F8
      03030303F8FF03030303030303030303030304020202020202040303030303030303
      0303F8030303030303F8FF0303030303030303040202 02FA0202020204030303
      0303030303F8FF0303F8FF030303F8FF03030303030303020202FA03FA020202
```

```
      040303030303030303F8FF03F803F8FF0303F8FF03030303030303FA02FA0303
      03FA0202020403030303030303F8FFF8030303F8FF0303F8FF03030303030303
      FA0303030303FA0202020403030303030303F80303030303F8FF0303F8FF0303
      0303030303030303030303FA0202020403030303030303030303030303F8FF03
      03F8FF0303030303030303030303030303FA0202020403030303030303030303
      0303F8FF0303F8FF0303030303030303030303030303FA0202020403030303030
      03030303030303F8FF0303F8FF0303030303030303030303030303FA0202020403
      0303030303030303030303030303F8FF0303F8FF0303030303030303030303030303FA
      0202040303030303030303030303030303F8FF03F8FF0303030303030303030
      03030303FA0202030303030303030303030303030303F8FFF803030303030303
      0303030303030303030303FA0303030303030303030303030303030303F803030303
      0303030303030303030303030303030303030303030303030303030303030303
      0303}
    Margin = 2
    NumGlyphs = 2
    Spacing = -1
    IsControl = True
  end
  object CancelBtn: TBitBtn
    Left = 108
    Top = 216
    Width = 69
    Height = 39
    Caption = '&Cancel'
    TabOrder = 4
    Kind = bkCancel
    Margin = 2
    Spacing = -1
    IsControl = True
  end
  object Panel2: TPanel
    Left = 16
    Top = 88
    Width = 153
    Height = 49
    BevelInner = bvLowered
    BevelOuter = bvNone
    TabOrder = 1
    object Label1: TLabel
      Left = 24
      Top = 8
      Width = 5
      Height = 13
    end
    object Label2: TLabel
      Left = 8
      Top = 8
      Width = 48
      Height = 13
      Caption = 'Value 2:'
    end
    object Edit2: TEdit
      Left = 8
      Top = 24
      Width = 121
      Height = 20
```

```
      MaxLength = 10
      TabOrder = 0
      Text = '0'
    end
  end
  object Panel1: TPanel
    Left = 16
    Top = 16
    Width = 153
    Height = 49
    BevelInner = bvLowered
    BevelOuter = bvNone
    TabOrder = 0
    object Label4: TLabel
      Left = 8
      Top = 8
      Width = 48
      Height = 13
      Caption = 'Value 1:'
    end
    object Edit1: TEdit
      Left = 8
      Top = 24
      Width = 121
      Height = 20
      MaxLength = 10
      TabOrder = 0
      Text = '0'
    end
  end
  object Panel3: TPanel
    Left = 16
    Top = 144
    Width = 153
    Height = 49
    BevelInner = bvLowered
    BevelOuter = bvNone
    TabOrder = 2
    object Label6: TLabel
      Left = 8
      Top = 8
      Width = 48
      Height = 13
      Caption = 'Value 3:'
    end
    object Edit3: TEdit
      Left = 8
      Top = 24
      Width = 121
      Height = 20
      MaxLength = 10
      TabOrder = 0
      Text = '0'
    end
  end
end
```

```
{ ***     END CODE FOR DDEDLG.DFM *** }

{ *** BEGIN CODE FOR DDEDLG.PAS *** }
{*******************************************************}
{                                                       }
{    Delphi 1.0 DDEML Demonstration Program             }
{    Copyright (c) 1996 by Borland International        }
{                                                       }
{*******************************************************}

{ This unit defines the interface to the DataEntry DDE
  server (DDEMLSRV.PAS).   It defines the Service, Topic,
  and Item names supported by the Server, and also defines
  a data structure which may be used by the Client to
  hold the sampled data locally.

  The Data Entry Server makes its data samples available
  in text (cf_Text) form as three separate Topics.   Clients
  may convert these into integer form for use with the
  data structure defined here.
}
unit Ddedlg;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, Buttons,
  StdCtrls, Mask, ExtCtrls;

type
  TDataEntry = class(TForm)
    OKBtn: TBitBtn;
    CancelBtn: TBitBtn;
    Bevel1: TBevel;
    Panel2: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Panel1: TPanel;
    Label4: TLabel;
    Panel3: TPanel;
    Label6: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure OKBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    S1, S2, S3: String;
    { Public declarations }
  end;

var
  DataEntry: TDataEntry;

implementation
```

```
{$R *.DFM}

procedure TDataEntry.OKBtnClick(Sender: TObject);
begin
    S1 := Edit1.Text;
    S2 := Edit2.Text;
    S3 := Edit3.Text;
end;

procedure TDataEntry.FormShow(Sender: TObject);
begin
    Edit1.Text := '0';
    Edit2.Text := '0';
    Edit3.Text := '0';
    Edit1.SetFocus;
end;

end.
{ ***     END CODE FOR DDEDLG.PAS *** }
```

TDBGrid and Multi-Selecting Records

NUMBER : 2976
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 3, 1997

TITLE : TDBGrid and Multi-Selecting Records

When you add [dgMultiSelect] to the Options property of a DBGrid, you give yourself the ability to select multiple records within the grid.

The records you select are represented as bookmarks and are stored in the SelectedRows property.

The SelectedRows property is an object of type TBookmarkList. The properties and methods are described below.

```
// property SelectedRows: TBookmarkList read FBookmarks;

//    TBookmarkList = class
//    public

       {* The Clear method will free all the selected records
          within the DBGrid *}
       // procedure Clear;

      {* The Delete method will delete all the selected rows
         from the dataset *}
      // procedure Delete;

      {* The Find method determines whether a bookmark is
         in the selected list. *}
      // function   Find(const Item: TBookmarkStr;
      //          var Index: Integer): Boolean;

      {* The IndexOf method returns the index of the
         bookmark within the Items property. *}
      // function IndexOf(const Item: TBookmarkStr): Integer;

      {* The Refresh method returns a boolean value to notify
         whether any orphans were dropped (deleted) during the
         time the record has been selected in the grid.   The
         refresh method can be used to update the selected list
         to minimize the possibility of accessing a deleted
         record. *}
      // function   Refresh: Boolean;   True = orphans found

      {* The Count property returns the number of currently
         selected items in the DBGrid *}
      // property Count: Integer read GetCount;
```

```
      {* The CurrentRowSelected property returns a boolean
         value and determines whether the current row is
         selected or not. *}
      // property CurrentRowSelected: Boolean
      //         read GetCurrentRowSelected
      //         write SetCurrentRowSelected;

      {* The Items property is a TStringList of
         TBookmarkStr *}
      // property Items[Index: Integer]: TBookmarkStr
      //         read GetItem; default;

//   end;

unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, DBGrids, DB, DBTables;

type
  TForm1 = class(TForm)
    Table1: TTable;
    DBGrid1: TDBGrid;
    Count: TButton;
    Selected: TButton;
    Clear: TButton;
    Delete: TButton;
    Select: TButton;
    GetBookMark: TButton;
    Find: TButton;
    FreeBookmark: TButton;
    DataSource1: TDataSource;
    procedure CountClick(Sender: TObject);
    procedure SelectedClick(Sender: TObject);
    procedure ClearClick(Sender: TObject);
    procedure DeleteClick(Sender: TObject);
    procedure SelectClick(Sender: TObject);
    procedure GetBookMarkClick(Sender: TObject);
    procedure FindClick(Sender: TObject);
    procedure FreeBookmarkClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  Bookmark1: TBookmark;
  z: Integer;

implementation
```

```
{$R *.DFM}

//Example of the Count property
procedure TForm1.CountClick(Sender: TObject);
begin
  if DBgrid1.SelectedRows.Count > 0 then
  begin
    showmessage(inttostr(DBgrid1.SelectedRows.Count));
  end;
end;

//Example of the CurrentRowSelected property
procedure TForm1.SelectedClick(Sender: TObject);
begin
  if DBgrid1.SelectedRows.CurrentRowSelected then
    showmessage('Selected');
end;

//Example of the Clear Method
procedure TForm1.ClearClick(Sender: TObject);
begin
  dbgrid1.SelectedRows.Clear;
end;

//Example of the Delete Method
procedure TForm1.DeleteClick(Sender: TObject);
begin
  DBgrid1.SelectedRows.Delete;
end;

{*
   This example iterates through the selected rows
   of the grid and displays the second field of
   the dataset.

   The Method DisableControls is used so that the
   DBGrid will not update when the dataset is changed.
   The last position of the dataset is saved as
   a TBookmark.

   The IndexOf method is called to check whether or
   not the bookmark is still existent.
   The decision of using the IndexOf method rather
   than the Refresh method should be determined by the
   specific application.
*}

procedure TForm1.SelectClick(Sender: TObject);
var
  x: word;
  TempBookmark: TBookMark;
begin
  DBGrid1.Datasource.Dataset.DisableControls;
  with DBgrid1.SelectedRows do
  if Count > 0 then
  begin
```

```
      TempBookmark:= DBGrid1.Datasource.Dataset.GetBookmark;
      for x:= 0 to Count - 1 do
      begin
        if IndexOf(Items[x]) > -1 then
        begin
          DBGrid1.Datasource.Dataset.Bookmark:= Items[x];
          showmessage(DBGrid1.Datasource.Dataset.Fields[1].AsString);
        end;
      end;
      DBGrid1.Datasource.Dataset.GotoBookmark(TempBookmark);
      DBGrid1.Datasource.Dataset.FreeBookmark(TempBookmark);
    end;
    DBGrid1.Datasource.Dataset.EnableControls;
end;

{*
This example allows you to set a bookmark and
and then search for the bookmarked record within
selected a record(s) within the DBGrid.
*}

//Sets a bookmark
procedure TForm1.GetBookMarkClick(Sender: TObject);
begin
  Bookmark1:= DBGrid1.Datasource.Dataset.GetBookmark;
end;

//Frees the bookmark
procedure TForm1.FreeBookmarkClick(Sender: TObject);
begin
  if assigned(Bookmark1) then
  begin
    DBGrid1.Datasource.Dataset.FreeBookmark(Bookmark1);
    Bookmark1:= nil;
  end;
end;

//Uses the Find method to locate the position of the
//bookmarked record within the selected list in the DBGrid
procedure TForm1.FindClick(Sender: TObject);
begin
  if assigned(Bookmark1) then
  begin
    if DBGrid1.SelectedRows.Find(TBookMarkStr(Bookmark1),z) then
      showmessage(inttostr(z));
  end;
end;

end.
```

Listing the field structures of a table.

NUMBER : 2977
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : May 21, 1996

TITLE : Listing the field structures of a table.

Q: How can I list the field structures of a table?

A: This project demonstrates listing the field structure from a given table, using the Fields and IndexDefs arrays, and displaying them in a listbox.

There is a demo (dbbrowsr.dpr) approaches this task differently. You may want to compare the two versions of this code.

Note: This code works with 16 bit only.

```
procedure TForm1.Button1Click(Sender: TObject); const
MyFielddefs: array[ftUnknown..ftGraphic] of string [8] =
  ('Unknown', 'String', 'Smallint', 'Integer', 'Word',
  'Boolean', 'Float', 'Currency', 'BCD', 'Date',
  'Time', 'DateTime', 'Bytes', 'VarBytes', 'Blob',
  'Memo', 'Graphic');

var
  i, Indx: integer;
  Definition: string;
begin
  for i := 0 to Table1.FieldCount - 1 do begin
    Definition := Table1.Fields[i].DisplayLabel;
    Definition := Definition + ' ' +
      MyFieldDefs[Table1.Fields[i].DataType];
    Table1.IndexDefs.Update;
    if Table1.Fields[i].IsIndexField then begin
      Indx := Table1.IndexDefs.Indexof(Table1.Fields[i].Name);
      if Indx > -1 then
        if ixPrimary in Table1.IndexDefs[Indx].Options then
          Definition := Definition + ' (Primary)';
    end;
    Listbox1.Items.Add(Definition);
  end;
end;
```

The version above does not work with the 32 bit version as there are more field types that must now be taken into account. Here is a version that works with the 32 bit version:

```
procedure TForm1.Button1Click(Sender: TObject);
const
  MyFielddefs: array[ftUnknown..ftTypedBinary] of string [11] =
```

```
      ('Unknown', 'String', 'Smallint', 'Integer',
      'Word', 'Boolean', 'Float', 'Currency', 'BCD',
      'Date', 'Time', 'DateTime', 'Bytes', 'VarBytes',
      'AutoInc', 'Blob', 'Memo', 'Graphic', 'FmtMemo',
      'ParadoxOle', 'DBaseOle', 'TypedBinary');

var
   i, Indx: integer;
   Definition: string;
begin
   for i := 0 to Table1.FieldCount - 1 do begin
      Definition := Table1.Fields[i].DisplayLabel;
      Definition := Definition + ' ' +
         MyFieldDefs[Table1.Fields[i].DataType];
      Table1.IndexDefs.Update;
      if Table1.Fields[i].IsIndexField then begin
         Indx := Table1.IndexDefs.Indexof(Table1.Fields[i].Name);
         if Indx > -1 then
            if ixPrimary in Table1.IndexDefs[Indx].Options then
               Definition := Definition + ' (Primary)';
      end;
      Listbox1.Items.Add(Definition);
   end;
end;
```

Showing deleted records in a dBASE table.

NUMBER : 2979
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : Showing deleted records in a dBASE table.

Q: How can I view dBASE records marked for deletion. That is, I want to view those records marked as "soft deletion"?

A: In a dBASE table, records are not removed from the table until the table is packed. Until that happens, records that are "deleted" are actually just marked as "to be" deleted. To show these existing but not displayed records, the following function, ShowDeleted(), makes use of a BDE API function, DbiSetProp(), to show records marked for deletion. It is not necessary to close and re-open the table when using this function. ShowDeleted() takes a TTable and a boolean variable as parameters. The boolean parameter determines whether or not to show deleted records.

Example code follows:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, DBCtrls, Grids, DBGrids,
  DB, DBTables;

type
  TForm1 = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    DBNavigator1: TDBNavigator;
    CheckBox1: TCheckBox;
    procedure CheckBox1Click(Sender: TObject);
  public
    procedure ShowDeleted(Table: TTable; ShowDeleted: Boolean);
  end;

var
  Form1: TForm1;

implementation

uses DBITYPES, DBIERRS, DBIPROCS;
```

```
{$R *.DFM}

procedure TForm1.ShowDeleted(Table: TTable; ShowDeleted: Boolean);
var
  rslt: DBIResult;
  szErrMsg: DBIMSG;
begin
  Table.DisableControls;
  try
    Check(DbiSetProp(hDBIObj(Table.Handle), curSOFTDELETEON,
      LongInt(ShowDeleted)));
  finally
    Table.EnableControls;
  end;
  Table.Refresh;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  ShowDeleted(Table1, CheckBox1.Checked);
end;

end.
```

Determining a memo's number of lines showing.

NUMBER : 2980
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : Determining a memo's number of lines showing.

Q: How can I tell how many lines a memo is capable of showing?

A: Here is the short and elegant way that works (most of the time):

```
function TForm1.MemoLinesShowing(memo: TMemo): integer;
var
  R: TRect;
begin
  Memo.Perform(EM_GETRECT, 0, Longint(@R));
  Result := (R.Bottom - R.Top) div Canvas.TextHeight('XXX');
end;
```

The problem with this code is that the TForm and the TMemo must both be using the same font. If the fonts are different, then the calculations are not accurate.

You have to retrieve the font height by selecting it into a device context. The reason you cannot use the font Height provided by Delphi is because Delphi caches the font infomation but doesn't acutally select the font into the DC (canvas) until it is actually going to draw something. This occurs in the painting event of the memo.

To get around this problem, you can get the memo's device context using the Windows API and get the font information from the device context to calculate the text height.

The function below illustrates this process:

```
function TForm1.MemoLinesShowingLong(Memo: TMemo): integer;
Var
  Oldfont: HFont;   {the old font}
  DC: THandle;      {Device context handle}
  i: integer;       {loop variable}
  Tm: TTextMetric; {text metric structure}
  TheRect: TRect;
begin
  DC := GetDC(Memo.Handle); {Get the memo's device context}
  try
   {Select the memo's font}
    OldFont := SelectObject(DC, Memo.Font.Handle);
    try
      GetTextMetrics(DC, Tm); {Get the text metric info}
      Memo.Perform(EM_GETRECT, 0, longint(@TheRect));
```

```
        Result := (TheRect.Bottom - TheRect.Top) div
            (Tm.tmHeight + Tm.tmExternalLeading);
      finally
        SelectObject(DC, Oldfont); {Select the old font}
      end;
    finally
      ReleaseDC(Memo.Handle, DC); {Release the device context}
    end;
end;
```

Delphi 2.0 Install Issues

NUMBER : 2981
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : May 21, 1996

TITLE : Delphi 2.0 Install Issues

Install Issues for Delphi 2.0
Last updated 3/13/96

1) Minimum System Requirements

Disk space:
a) OCX, DLL, and licensing files will require approx. 10 megs in your \windows\system directory.

b) Expansion of files during installation temporarily requires 10-15 megs in the drive that contains your \temp directory. For example, if you are installing Delphi 2.0 to drive D:, and your temporary directory path is C:\WINDOWS\TEMP, so that drive C will require 10 - 15 megs during file expansion.

Minimum system requirements:

Processesor
Desktop : intel 386dx based pc or higher with math coprocessor
Developer: intel 486/25 based pc or higher
C/S Suite: intel 486/25 based pc or higher
MS windows '95 or NT 3.51
8 meg memory (12 recommended)
50 meg hard disk space
CD rom drive (3.5 floppy disks available seperately)
mouse or Windows pointing device

The chart below shows disk space used resulting from sample installs run on a Windows '95 system.

"Permananet Megs Used" represents the amount of disk used after a sample install to a stock Window's '95 system with no other applications running. These figures are approximate. Your disk cluster size may cause the file sizes to vary. Also, Window's '95 handling of swap file may vary according to your system configuration.

Product	Permanent Megs Used	\Borland	\Windows	(Swap \System File)
CSS compact	45	39	6	9
CSS full	105	98	7	24
Dev compact	43	37	6	9
Dev full	104	97	7	20
Desk compact	32	30	2	9
Desk full	61	59	2	15

2)
a) Canceling the Delphi 2.0 install
Canceling before 10 - 15% may not clean the Window's '95 registry. This appears to be a limitation of the Install Shield engine.

Canceling the install after 15% will cause Delphi 2.0 to be un-installed.

b) Canceling Delphi 2.0 Online Registration
If you wish to register Delphi 2.0 at a later time, you may cancel the Registration Wizard. Canceling the Registration Wizard will not cause an un-install. You may register Delphi 2.0 later from the "Delphi 2.0 Online Registration" icon located in the Borland Delphi 2.0 directory.

3) Un-Installing
Files created after the install of Delphi 2.0 and the directories that contain them, will not be deleted. For example, if you were to create a new project in \Borland\Delphi 2.0\Bin, the un-install would leave your project files and the \Borland\Delphi 2.0\Bin directory that contained them.

A general rule of thumb is that only files from the most recent install will be un-installed.

Files marked as shared, OCX's and DLL's for example, that have a "use" count of zero will cause a delete confirmation dialog to appear for each file. Microsoft standards for un-installing do not include a "Delete All" option at this stage.

4) Re-installing Delphi 2.0 on Window's 95
If an earlier installation of Delphi 2.0 already

exists on your Window's 95 system, you must run the un-install program. This is necessary to cleanup the system registry. The un-install utility is located off the Start menu, Settings|Control Panel|Add/Remove Programs.

5) Installing to systems with Paradox 7.0
The necessary order to install both Delphi 2.0 and Paradox 7.0 is to

a) install Paradox 7.0
b) install Delphi 2.0.

If Delphi 2.0 already exists on the system, un-install Delphi 2.0, then do a) and b) listed above. If you un-install either Delphi 2.0 or Paradox 7.0, you will need to re-install the 32 bit IDAPI.

6) Installing Delphi 1.0 and 2.0 on the same system
Although Delphi 1.0 and 2.0 will run on the same system, don't combine Delphi 1.0 and 2.0 into the same directory. Installing to the same directory will cause difficulty when the Delphi IDE attempts to locate the Help file.

7) Errors During the Install

Error 101
Error 112;
There is not enough room in the \Temp directory to expand files. Expansion of files during installation temporarily requires 10-15 megs in the drive that contains your \temp directory. For example, if you are installing Delphi 2.0 to drive D:, and your temporary directory path is C:\WINDOWS\TEMP, so that drive C will require 10 - 15 megs during file expansion.

"Cannot install Delphi, Local Interbase is Running"
Shut down Interbase from the system tray by right clicking the Interbase icon. The System Tray is the indentation located on right hand side of the task bar)

Registration Wizard dialog appears as a blank
InstallShield could not find the install script.
Cancel the dialog and run the Registration Wizard from the icon "Delphi 2.0 Online Registration".

Temporary directories remain after install to NT.
On NT systems temporary directories named \ISTEMP0x may be left over. These directories will probably be empty, and may be deleted.

How to Validate ISBNs

NUMBER : 2988
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : How to Validate ISBNs

ISBNs (or International Standard Book Numbers) are mystical code numbers that uniquely identify books. The purpose of this article is to remove the mystery surrounding the structure of the ISBN, allowing applications to perform data validation on entered candidate ISBNs.

ISBNs are composed of thirteen characters, limited to the number characters "0" through "9", the hyphen, and the letter "X". This thirteen-character code is divided into four parts, each separated by hyphens: group identifier, publisher identifier, book identification for the publisher, and the check digit. The first part (group identifier) is used to identify countries, geographical regions, languages, etc. The second part (publisher identifier) uniquely identifies the publisher. The third part (book identifier) uniquely identifies a given book within a publisher's collection. The fourth and final part (check digit) is used with the other digits in the code in an algorithm to derive a verifiable ISBN. The number of digits in the first three parts of an ISBN may contain a variable number of digits, but the check digit will always consist of a single character (between "0" and "9", or "X" for a value of 10) and the ISBN as a whole will always consists of thirteen characters (ten numbers plus the three hyphens dividing the four parts of the ISBN).

The ISBN 3-88053-002-5 breaks down into the parts:

Group: 3
Publisher: 88053
Book: 002
Check Digit: 5

An ISBN can be verified to be a valid code using a simple mathematical algorithm. This algorithm takes each of the nine single digits from the first three parts if the ISBN (sans the non-numeric hyphens), multiplies each single digit by a number that is less than eleven the number of positions from the left each digit that is in the ISBN, adds together the result of each multiplication plus the check digit, and then divides that number by eleven. If that division by eleven results in no remainder (i.e., the number is modulo 11), the candidate ISBN is a valid ISBN. For example, using the previous sample ISBN 3-88053-002-5:

ISBN: 3 8 8 0 5 3 0 0 2 5
Digit Multiplier: 10 9 8 7 6 5 4 3 2 1
Product: 30+72+64+00+30+15+00+00+04+05 = 220

Since 220 is evenly divisible by eleven, this candidate IDBN is a valid ISBN code.

This verification algorithm is easily translated into Pascal/Delphi code. String manipulation functions and procedures are used to extract the check digit and the remainder of the ISBN from the String type value passed to a validation function. The check digit is converted to Integer type, which forms the start value of the aggregate variable onto which the multiplication of each digit in the remainder of the ISBN (the single digits that comprise the first three parts of the ISBN) will be added. A For loop is used to sequentially process each digit in the remainder, ignoring the hyphens, multiplying each digit times its position in the ISBN remainder relative to the other digits in the remainder. The final value of this aggregate variable is then checked to see whether it is evenly divisible by eleven (indicating a valid ISBN) or not (indicating an invalid candidate ISBN).

Here is an example of this methodology applied in a Delphi function:

```
function IsISBN(ISBN: String): Boolean;
var
  Number, CheckDigit: String;
  CheckValue, CheckSum, Err: Integer;
  i, Cnt: Word;
begin
  {Get check digit}
  CheckDigit := Copy(ISBN, Length(ISBN), 1);
  {Get rest of ISBN, minus check digit and its hyphen}
  Number := Copy(ISBN, 1, Length(ISBN) - 2);
  {Length of ISBN remainder must be 11 and check digit between 9 and 9 or
   X}
  if (Length(Number) = 11) and (Pos(CheckDigit, '0123456789X') > 0) then
    begin
    {Get numeric value for check digit}
    if (CheckDigit = 'X') then
      CheckSum := 10
    else
      Val(CheckDigit, CheckSum, Err);
    {Iterate through ISBN remainder, applying decode algorithm}
    Cnt := 1;
    for i := 1 to 12 do begin
      {Act only if current character is between "0" and "9" to exclude
       hyphens}
      if (Pos(Number[i], '0123456789') > 0) then begin
        Val(Number[i], CheckValue, Err);
        {Algorithm for each character in ISBN remainder, Cnt is the nth
        character so processed}
        CheckSum := CheckSum + CheckValue * (11 - Cnt);
        Inc(Cnt);
      end;
    end;
    {Verify final value is evenly divisible by 11}
    if (CheckSum MOD 11 = 0) then
      IsISBN := True
    else
      IsISBN := False;
  end
  else
    IsISBN := False;
```

```
end;
```

This is a simplified example, kept simple to best demonstrate the algorithm to decode ISBNs. There are a number of additional features that would be desirable to add for use in a real-world application. For instance, this example function requires the candidate ISBN be passed as a Pascal String type value, with the hyphens dividing the four parts of the ISBN. Added functionality might accommodate evaluating candidate ISBNs entered without the hyphens. Another feature that might be added is checking that ensures three hyphens are properly included, as opposed to just thirteen number characters.

BDE setup for Peer-To-Peer(Non-Dedicated) Networks

NUMBER : 2989
PRODUCT : BDE
VERSION : 3.0
OS : Windows
DATE : June 24, 1996

TITLE : BDE setup for Peer-To-Peer(Non-Dedicated) Networks

Using a BDE32 Application on a Peer-To-Peer Network

A Peer-To-Peer network (a network where each machine acts as a client and a server) can be one of the following, including other network platforms that are compatible with these:

1) Windows 95
2) Windows NT
3) Lantastic
4) Netware Lite

The BDE automatically detects when tables reside on a network drive, but it cannot detect whether the tables are on a dedicated server or a server/client. Dedicated servers notify client applications that a file has been modified or locked. This functionality is not present in Peer-To-Peer (non-dedicated) networks. To achieve this functionality with Peer-To-Peer networks set "LOCAL SHARE" to TRUE in the BDE Configuration Utility on the System page. This must be done on all BDE clients that access the tables on networks listed above. This is not necessary for Novell File Server type networks.

If the tables that are being used are Paradox, there must also be a directory used for network control. This directory must also reside on the network for all client applications to use. It is good practice to have a separate directory for the application, network, and tables. The following is an example:

```
(Shared Directory)
        |
        |--- (Tables Directory)
        |--- (EXE Directory)
        |--- (Network Directory)
```

There are two different BDE environments that must also be considered:

1) Using only BDE 32Bit applications.
2) Using BDE 32Bit applications along with BDE 16Bit applications.

Setup for 32Bit Only Applications

The 32Bit BDE fully supports the UNC naming convention along with long file names. It is recommended that the UNC convention is used for all BDE network connections. UNC removes the need for mapped drives. This will allow access to the tables and network directory without the user being mapped to the drive. UNC has the following syntax:

\\(server name)\(share name)\(path)+(file name)

Here is a simple example of a standard BDE alias using UNC:

Alias: MyUNCAlias
Type: STANDARD
Path: \\FooServer\FooShare\Sharedir\Tables
Default Driver: Paradox

The network directory can be setup in the same fashion:
Drivers: Paradox
Net Dir: \\FooServer\FooShare\Sharedir\NetDir

The network directory can be set at runtime using session.netfiledir (Delphi) or DbiSetProp (C++ / Delphi)

If for some reason UNC cannot be used with the 32Bit application, follow directions for using BDE 32Bit and 16Bit applications.

Setup for 16Bit and 32Bit BDE Applications

Since the 16Bit Windows API does not support UNC, neither does the 16Bit BDE. To allow applications to share the tables, all clients must be mapped to the same directory on the server. If the server is also used as a client, all other clients must be mapped to the root of the drive. Drive letters from client to client do not have to be identical. Here are some examples of what will and will not work:

Client1:
Path: X:\ShareDir\Tables
Client2:
Path: X:\ShareDir\Tables
This is OK

Client1: (Also the machine with the tables):
Path: C:\ShareDir\Tables
Client2:
Path: X:\ShareDir\Tables
This is OK

Client1: (Also the machine with the tables):
    Path: C:\ShareDir\Tables
Client2:
    Path: X:\ShareDir\Tables
Client3:
    Path: R:\ShareDir\Tables
This is OK

Client1:
    Path: X:\ShareDir\Tables
Client2:
    Path: X:\Tables (Where X:\Tables is actually
X:\ShareDir\Tables, but shared on the ShareDir directory)
  This will not work. The BDE must be able to make the
same entry into the Network Control file.

In Summary (setup for Peer-To-Peer networks):
---
16 and / or 32Bit Applications:
1) Turn "LOCAL SHARE" to TRUE in the BDE Configuration Utility.
2) Do not use the UNC naming convention.
3) Do not use tables with long file names.
4) Make sure that all clients are mapped to the same directory on the server.

32Bit Only Applications:
1) Turn "LOCAL SHARE" to TRUE in the BDE Configuration Utility
2) Use the UNC naming convention to achieve a path to the network directory and table directory.

If the above steps are not followed, users could be
locked out of the tables getting error:
"Directory is controlled by other .NET file."
"File: (Path1) PDOXUSRS.LCK"
"Directory: (Path2)"
    OR
"Multiple .NET files in use."
"File: (Path) PDOXUSRS.LCK"

Removing "Lock file has grown too large" Error

NUMBER : 2993
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 5, 1996

TITLE : Removing "Lock file has grown too large" Error

Lock file has grown too large, Decimal:9495 Hex:2517

This problem is specific to Paradox tables and can be caused in any BDE (16 or 32 Bit) application that meet some or all of the following criteria:

1) The Executable is in the same directory as the table.
2) The Private Directory is not correctly set or not set at all.
3a) Delphi: Having a TTable open on a paradox table and then performing multiple TQuery operations.
3b) C / C++: Having a table open with DbiOpenTable and then performing multiple queries with DbiQExec and/or DbiQExecDirect.
4) LOCAL SHARE set to true in the BDE Configuration Utility on the System page.

To solve the problem, make sure that your application has done ALL of the following:

1) Under the directory where the executable is, create three new directories: TABLES, PRIV, and NET. Place all the tables for the application into the TABLES directory.
2) Set the session's private directory to the PRIV directory. Take the following steps according to the software you are using.

   DELPHI:
   ```
   Session.PrivateDir := ExtractFilePath(ParamStr(0)) + 'PRIV';
   ```

   C / C++:
   ```
   DbiSetPrivateDir(szPath);
   // szPath is the fully qualified path (not relative)
   // to the PRIV directory.
   ```

3) Set the session's network directory to the NET directory. Take the following steps according to the software you are using.

   DELPHI:
   ```
   Session.NetFileDir := ExtractFilePath(ParamStr(0)) + 'NET';
   ```

   C / C++:

```
DbiSetProp(hSes, sesNETFILE, (UINT32)szPath);
// szPath is the fully qualified path (not relative)
// to the NET directory.

// hSes is the current session handle.   This can be
// retrieved using the DBiGetCurrSession function.
```

4) If LOCAL SHARE is set to true and you are not sharing tables between different applications at the same time, change LOCAL SHARE to false.

The above steps will correct the Lock File Too Large error.

SQL: Using the SUBSTRING Function

NUMBER : 2962
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 14, 1996

TITLE : SQL: Using the SUBSTRING Function

The SQL function SUBSTRING can be used in Delphi applications that include local SQL queries, but is not supported for InterBase (IB) or the Local InterBase Server (LIBS) tables. What follows is the syntax for the SUBSTRING function, examples of its use in local SQL queries, and an alternative that will return the same results for IB/LIBS tables.

The syntax for the SUBSTRING function is:

```
SUBSTRING(<column> FROM <start> [, FOR <length>])
```

Where:

<column> is the name of the column in the table from which the sub-string is to be extracted.

<start> is the point in the column value from which the sub-string to be extracted will start.

<length> is the length of the sub-string to be extracted.

Using these values, the use of the SUBSTRING function below would return the second, third, and fourth characters from a column named COMPANY:

```
SUBSTRING(COMPANY FROM 2 FOR 3)
```

The SUBSTRING function can be used either in the field list for a SELECT query or in the WHERE clause of a query to allow for comparing a value with a specific sub-set of a column. The SUBSTRING function can only be used with String type columns (the CHAR type in SQL parlance). Here is an example of the SUBSTRING function used in a columns list in a SELECT query (using the sample Paradox table CUSTOMER.DB):

```
SELECT (SUBSTRING(C."COMPANY" FROM 1 FOR 3)) AS SS
FROM "CUSTOMER.DB" C
```

This SQL query extracts the first three characters from the COMPANY column, returning them as the calculated column named SS. Now, an example of the SUBSTRING function used in the WHERE clause of an SQL query (using the same sample table):

```
SELECT C."COMPANY"
FROM "CUSTOMER.DB" C
WHERE SUBSTRING(C."COMPANY" FROM 2 FOR 2) = "an"
```

This query returns all rows from the table where the second and third

characters in the COMPANY column are "ar".

As the SUBSTRING function is not supported at all by IB or LIBS databases, it is not possible to have a sub-string operation in the column list of a query (exception: IB can do sub-strings via User-Defined Functions). But through use of the LIKE operator and the accompanying character substitution marker, it is possible to effect a sub-string in a WHERE clause. For example, using the sample table EMPLOYEE (in the EMPLOYEE.GDB database):

```
SELECT LAST_NAME, FIRST_NAME
FROM EMPLOYEE
WHERE LAST_NAME LIKE "_an%"
```

This SQL query would return all rows in the table where the second and third characters of the LAST_NAME column are "an", similar to the previous example for the Paradox table. While IB and LIBS databases would require this method for performing sub-string comparisons in the WHERE clause of a query and cannot use the SUBSTRING function, Paradox and dBASE tables (i.e., local SQL) can use either method.

Delphi Client/Server Suite 2.0 for Windows 95 & NTDelphi Client/Server Suite 2.0 for Windows 95 & NT

NUMBER : 3003
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : May 17, 1996

TITLE : Delphi Client/Server Suite 2.0 for Windows 95 & NT

Delphi Client/Server Suite 2.0 for Windows 95 and NT
Quick Info Guide

Delphi Client/Server Suite 2.0 provides you with all the tools you need to build scalable client and server applications while shortening every stage of the development cycle everything you need to boost productivity and gain a competitive advantage.

Scalable client/server architecture

* Create fast, standalone, royalty-free scalable applications that are up to 15 to 50 times faster than p-code interpreters.
* Unleash the power of Windows 95 and Windows NT with complete support for OCXs, OLE Automation, multi-threading, Unicode, MAPI, and more.
* Increase productivity with more than 100 reusable components.
* Deploy royalty-free applications using the NEW! 32-bit SQL Links native drivers for Oracle, Sybase, MS SQL Server, and InterBase.(R)
* Visually manage server-specific meta data like stored procedures and triggers with the NEW! SQL Database Explorer.
* Test, debug, and tune SQL applications with the NEW! SQL Monitor.
* Increase server responsiveness with NEW! Cached Updates.
* Build and test multiuser SQL applications with the NEW! InterBase NT (2-user license).
* Use the NEW! Data Module Objects to separate business rules from application code, and scalable NEW! Data Dictionary to implement and maintain data integrity.
* Generate bug-free ANSI SQL-92 code with the Visual Query Builder.
* Manage complex team projects with the NEW! integrated PVCS Version Control.

Everything you need to gain a competitive advantage!

* High-performance, optimizing 32-bit native-code compiler
* Create reusable DLLs and royalty-free, standalone EXEs
* Full Windows 95 support for OCX controls, and OLE Automation controllers and servers
* Complete access to Windows 95 APIs for multi-threading, Unicode, MAPI, and more

* Complete suite of Windows 95 components
* 32-bit SQL Links native drivers, with unlimited deployment license for Oracle, Sybase, MS SQL Server, and InterBase
* InterBase NT (2-user lic.) for scalable multiuser SQL development
* SQL Database Explorer to visually manage server-specific meta data
* SQL Monitor to test, debug, and tune SQL applications
* Integrated InterSolv PVCS Version Control for team development
* Visual Query Builder to generate bug-free SQL code
* Cached Updates to speed up server response time
* Data Pump Expert for rapid upsizing and application scaling
* Object-oriented component architecture
* Object Repository for storing and reusing objects, business rules, and forms
* Visual Form Inheritance to reduce coding and simplify maintenance
* Integrated Development Environment
* Visual Component Library (VCL) with more than 100 reusable components
* VCL Source Code and manual to develop or customize components
* Sophisticated data-aware components
* 32-bit Borland(R) Database Engine with low-level API support and Help files
* Database Explorer to visually browse and modify tables and aliases
* Data Module Objects to separate business rules from application code for easier maintenance
* Scalable Data Dictionary to implement and maintain data integrity
* Complete ODBC connectivity
* 32-bit Local InterBase Server for off-line SQL development
* 32-bit ReportSmith(R) for sophisticated reporting
* InstallShield(R) Express for building professional installs
* WinSight(TM) to monitor Windows messages
* Expanded Open Tools API to integrate your favorite tools
* Complete documentation including Object Pascal Language Guide
* Includes 16-bit Delphi 1.0 for Windows 3.1 development

Delphi Facts	Delphi	PowerBuilder	Visual Basic
Optimizing 32-bit native-code compiler	Y	N	N
Create standalone EXEs and DLLs	Y	N	N
Create multi-threaded Windows 95/NT applications	Y	N	N
Full support for OLE Automation and OCXs	Y	N	Y
Includes more than 100 reusable components	Y	N	N
Easily build or customize components	Y	N	N
High-performance Visual Form	Y	N	N

Inheritance			
Object Repository for forms and Data Modules	Y	N	N
Scalable Data Dictionary	Y	N	N
Fully scalable client/server architecture	Y	Y	N
Fast learning curve for increased productivity	Y	N	Y
SQL Database Explorer to visually manage meta data	Y	N	N
SQL Monitor to test, debug, and true SQL apps	Y	N	N

Minimum System Requirements

* Intel 486/25-based PC or higher
* Microsoft Windows 95 or Windows NT 3.51
* 8Mb of memory (12Mb recommended)
* 50Mb hard disk space
* CD-ROM drive
* Mouse or other Windows pointing device

Borland Delphi Client/Server Suite support services

* Fast Fax for Detailed Information: 1-800-408-0001
* TechFax(TM) for Technical Information: 1-800-822-4269
* Connections Developer Program: 1-800-353-2211
* Free Install Support: (408) 461-9195
* Credit Card Advisor Line: 1-800-330-3372
* 900-Advisor Lines:
    - Delphi and Delphi Client/Server: 1-900-555-1015
    - Local InterBase Server: 1-900-555-1013
    - ReportSmith: 1-900-555-1011
    - For Technical Support Service contracts and information: 1-800-523-7070

Satisfaction guaranteed!

You can buy the Delphi Client/Server Suite with complete assurance. If for any reason you are not fully satisfied with your purchase, you can return it to Borland within 90 days. No questions asked!

Authorization required to sell these products. Call 1-800-408-0001 to request a client/server authorization form. Educational pricing is available through Borland Authorized Educational Resellers. Call 1-800-847-7797. Advisor Line charges: $2.00 per minute, first minute free.

Borland's DELPHI products and services are not associated with or sponsored by Delphi Internet, an online service and Internet access provider.

subject to the terms of the No-Nonsense License Statement that you received with the Borland product to which this information pertains.

Performing database queries in a background thread

NUMBER : 3005
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 13, 1996

TITLE : Performing database queries in a background thread

This document explains how to perform queries in a background thread by using the TThread class. For information on the general usage of the TThread class, please refer to the Borland documentation and to the online help. You should be aware of how to use Delphi 2.0's database components to understand the this TI's contents.

Two requirements must be met in order to perform a threaded query. First, the query to be threaded must be contained within its own session by using a separate TSession component. Therefore, you would place a TSession component on your form and assign it's name to the SessonName property of the TQuery component to be used in the thread. You must use a separate TSession component for each TQuery component to be used in a thread. If you are also using a TDataBase component, a separate TDataBase must be used for each threaded query as well. The second requirement is that the threaded TQuery component must not be connected to a TDataSource in the context of the thread in which it will be executed. This must be done in the context of the primary thread.

The code example below illustrates this process. This unit shows a form which contains two each of the following comopnents: TSession, TDatabase, TQuery, TDataSource and TDBGrid. These components have the following property settings:

```
Session1
        Active   True;
        SessionName   "Ses1"

DataBase1
        AliasName       "IBLOCAL"
        DatabaseName "DB1"
        SessionName   "Ses1"

Query1
        DataBaseName"DB1"
        SessionName   "Ses1"
        SQL.Strings     "Select * from employee"

DataSource1
        DataSet         ""
```

```
DBGrid1
        DataSource      DataSource1

Session2
        Active   True;
        SessionName   "Ses2"

DataBase2
        AliasName       "IBLOCAL"
        DatabaseName "DB2"
        SessionName   "Ses2"

Query2
        DataBaseName"DB2"
        SessionName   "Ses2"
        SQL.Strings     "Select * from customer"

DataSource2
        DataSet         ""

DBGrid1
        DataSource      DataSource2
```

Notice that the DataSet property for both TDataSource components do not refer to anything initially. This will be set at run-time as illustrated in the code.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Grids, DBGrids, DB, DBTables;

type

  TForm1 = class(TForm)
    Session1: TSession;
    Session2: TSession;
    Database1: TDatabase;
    Database2: TDatabase;
    Query1: TQuery;
    Query2: TQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    GoBtn1: TButton;
    procedure GoBtn1Click(Sender: TObject);
  end;

  TQueryThread = class(TThread)
  private
```

```
    FSession: TSession;
    FDatabase: TDataBase;
    FQuery: TQuery;
    FDatasource: TDatasource;
    FQueryException: Exception;
    procedure ConnectDataSource;
    procedure ShowQryError;
  protected
    procedure Execute; override;
  public
    constructor Create(Session: TSession; DataBase:
      TDatabase; Query: TQuery; DataSource: TDataSource);
      virtual;
  end;


var
  Form1: TForm1;

implementation

constructor TQueryThread.Create(Session: TSession; DataBase:
  TDatabase; Query: TQuery; Datasource: TDataSource);
begin
  inherited Create(True);      // Create thread in a
suspendend state
  FSession := Session;          // connect all private fields
  FDatabase := DataBase;
  FQuery := Query;
  FDataSource := Datasource;
  FreeOnTerminate := True;     // Have thread object free
itself when terminated
  Resume;                       // Resume thread execution
end;

procedure TQueryThread.Execute;
begin
  try
    { Run the query and connect the datasource to the TQuery
      component by calling ConnectDataSource from main
      thread (Synchronize used for this purpose)}
    FQuery.Open;
    Synchronize(ConnectDataSource);
  except
    { Capture exception, if one occurs, and handle it in the
      context of the main thread (Synchonize used for this
      purpose. }
    FQueryException := ExceptObject as Exception;
    Synchronize(ShowQryError);
  end;
end;

procedure TQueryThread.ConnectDataSource;
begin
  FDataSource.DataSet := FQuery;   // Connect the DataSource
to the TQuery
```

```
end;

procedure TQueryThread.ShowQryError;
begin
  Application.ShowException(FQueryException); // Handle the
exception
end;

procedure RunBackgroundQuery(Session: TSession; DataBase:
TDataBase;
                             Query: TQuery; DataSource:
TDataSource);
begin
  { Create a TThread instance with the various parameters. }
  TQueryThread.Create(Session, Database, Query, DataSource);
end;


{$R *.DFM}

procedure TForm1.GoBtn1Click(Sender: TObject);
begin
  { Run two separate queries, each in their own thread }
  RunBackgroundQuery(Session1, DataBase1, Query1,
Datasource1);
  RunBackgroundQuery(Session2, DataBase2, Query2,
Datasource2);
end;

end.
```

The TForm1.GoBtn1Click method is an event handle for a button click event. This event handler calls the RunBackgroundQuery procedure twice, each time passing a different set of database components. RunBackgroundQuery creates a separate instance of the TQueryThread class, passing the various database components to its constructor which in turn assigns them to the appropriate TQueryThread private data fields.

The TQueryThread contains two user-defined procedures: ConnectDataSource and ShowQryError. ConnectDataSource connects FDataSource.DataSet to FQuery. However, it does this in the primary thread by using the TThread.Synchronize method. ShowQryError handles the exception in the context of the primary thread, again by using the Synchronize method. The Create constructor and Execute method are explained in the code's comments.

Obtaining the Physical Path of a Table

NUMBER : 3100
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 20, 1996

TITLE : Obtaining the Physical Path of a Table

TITLE: Obtaining The Physical Path of a Table
Submitted: August 13, 1996
Author: Xavier Pacheco

When a Table is referenced through an alias, the physical path is not readily available. To obtain this path, use the DbiGetDatabaseDesc BDE function. This function takes the alias name and a pointer to a DBDesc structure. The DBDesc structure will be filled with the information pertaining to that alias. This structure is defined as:

```
pDBDesc = ^DBDesc;
DBDesc = packed record              { A given Database Description }
  szName              : DBINAME;    { Logical name (Or alias) }
  szText              : DBINAME;    { Descriptive text }
  szPhyName           : DBIPATH;    { Physical name/path }
  szDbType            : DBINAME;    { Database type }
end;
```

The physical name/path will be contained in the szPhyName field of the DBDesc structure.

Possible return values for the DBIGetDatbaseDesc function are:

DBIERR_NONE	The database description for pszName was retrieved successfully.
DBIERR_OBJNOTFOUND	The database named in pszName was not found.

The code example below illustrates how to obtain the physical path name of a TTable component using the DBDemos alias:

```
var
  vDBDesc: DBDesc;
  DirTable: String;
begin
  Check(DbiGetDatabaseDesc(PChar(Table1.DatabaseName), @vDBDesc));
  DirTable := Format('%s\%s', [vDBDesc.szPhyName, Table1.TableName]);
  ShowMessage(DirTable);
end;
```

TForm.MDIChildren[] Array and Form Creation

NUMBER : 3050
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 13, 1996

TITLE : TForm.MDIChildren[] Array and Form Creation

The documentation of TForm.MDIChildren[] states that the index of the first-created MDI child is 0. This is incorrect -- the index of the most-recently-created MDI child is always 0, and the index of the first-created MDI child is always MDIChildCount - 1.

With this in mind, you can use the following code to iterate over the MDI child array from from the first-created to the last:

```
procedure TForm1.IterateOverMDIChildren;
var
  i: integer;
begin
  for i := MDIChildCount - 1 downto 0 do begin
    { do something with MDI child here }
  end;
end;
```

Delphi Client/Server Certification Program

NUMBER : 3051
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 12, 1996

TITLE : Delphi Client/Server Certification Program

Borland International

Delphi Client/Server Certification Program

The Delphi Client/Server Certification Program was developed by Borland to meet the needs of IS departments, developers, and training professionals. Certification is a key accomplishment earned by an alliance of organizations and individuals devoted to the ongoing delivery of training and of quality business solutions built with Delphi Client/Server. This alliance enables Borland partners to deliver distinctive applications and education to our customers worldwide.

Certification Tracts

Through certification, Borland validates an individual's Delphi Client/Server knowledge and skill. The validation is earned by meeting requirements in one of two tracts:

1) Certified Delphi Client/Server Developer
2) Certified Delphi Client/Server Training Professional

Borland's certified professionals enjoy the important status required to develop and train in America's most demanding corporate development environments. In addition, Borland certified developers and trainers:

- receive a Certificate of Recognition
- are entitled to use the Borland Delphi C/S Certified logo which designates their expertise with Delphi Client/Server
- are entitled to purchase Borland Delphi Client/Server 2.0 courseware (Training Professionals only)
- receive a free listing on Borland's On-line Resource Locator*

Requirements for Certified Delphi C/S Developers

To become a Certified Delphi C/S Developer, you must:

- complete the Borland Delphi Client/Server Exam with a passing score
- sign a Delphi C/S Developer Certification Agreement

Requirements for Certified Delphi C/S Training Professionals

To achieve the status and distinction of becoming a Certified Delphi Client/Server Training Professional, you must:
- complete the Borland Delphi Client/Server Exam with a passing score
- successfully complete a Borland-sponsored Delphi Client/Server 2.0 Train-the-Trainer course
- sign a Delphi Client/Server Training Professional Certification Agreement

Certification Renewal

Certification renewal is required annually or in conjunction with major new releases of Delphi Client/Server Suite. To maintain continuous Certification status, you must renew certification within 60 days of release of new certification testing. Renewal may include passing another certification exam, additional fees, and attendance at additional Borland authorized training courses.

Following are details and instructions on each of the requirements for certification.

- complete the Borland Delphi C/S Certification Exam with a passing score

  A certified professional must have exceptional Delphi C/S skills and knowledge in order to assist others in the full use the product. Individuals who are seeking the status of Delphi C/S certification can make two attemtps at passing the exam. If both attempts are unsuccessful, you must attend a Delphi C/S training course taught by a certified instructor before making a third attempt to pass the exam. For information on Delphi Client/Server product training send an email to kalderman@wpo.borland.com.

  Preparing for the Delphi Client/Server Exam:

  Individuals may download the Study Guide (http://www.borland.com) which outlines the objectives used to create the questions on the Delphi Client/Server Exam. Candidates are also encouraged to take advantage of any Delphi Client/Server 2.0 classes taught by Certified Delphi Client/Server Training Professionals and study Borland Press books.

  Testing Centers and Costs:

  Price: $140/exam
  (Currently testing is available in the United State and Canada. Watch this space as we bring additional testing centers online internationally.

All testing is administered by Sylvan Prometric testing centers. Sylvan Prometric has over 200 testing centers worldwide. You are allowed 75 minutes to complete the Delphi C/S exam. Your score will be presented at the conclusion of the exam. To register to take the Delphi Client/Server Exam in a Sylvan testing center near you, call 1-800-430-EXAM (1-800-430-3926).

- successfully complete a Borland Delphi Client/Server 2.0 Train-the-Trainer course

  The Delphi C/S 2.0 Train-the-Trainer class is required by all Training Professionals seeking certification. This five-day class of in-depth courseware instruction offers instructor discussions, teachbacks, reviews and tips of Borland's official Delphi Client/Server 2.0 courseware. Included in the $2995 price is one Delphi Client/Server Instructor Guide, ten Delphi Client/Server Student Guides and five free support phone calls to Borland's courseware developer all valued at over $2800.

  For specific dates and locations of the upcoming Train-the-Trainer classes, please contact Lisa Coenen at 408-431-5758 or by email at lcoenen@wpo.borland.com.

- sign a Delphi C/S Developer or Training Professional Certification Agreement

  Once Borland receives your exam results from Sylvan Prometric, you will be notified. A Certification Agreement will be mailed to you for signature. When the signed original agreement is returned to Borland, you will receive official notification of certification along with the certifiation logo and Certificate of Recognition. A copy of the Certification Agreement is downloaded with this document.

Note: Certification qualifications may vary slightly by country. For details on certification programs outside the United States and Canada contact your local Borland sales office. *Free listings are available to members of Borland Connections, Premier Partners and Authorized Education centers.

Authorized Client/Server Education Centers

NUMBER : 3052
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 12, 1996

TITLE : Authorized Client/Server Education Centers

Borland International

Authorized Client/Service Education Centers

Borland Authorized Client/Server Education Centers program includes organizations whose training expertise and environment meet the high technical and training standards set by Borland. Our purpose is to create an alliance with organizations that provide professional quality and value-added services necessary to meet the technical and educational needs of Borland's client/server customers worldwide.

Benefits of Borland Authorized Client/Service Education Centers Status

In addition to the prestige of attaining the status of a Borland Authorized Client/Server Education Center, recognized organizations enjoy the following benefits:

- Preferred pricing of Borland's Delphi Client/Server 2.0 courseware
- Ability to deliver Borland Delphi Client/Server 2.0 Training Bundles

Authorization Requirements

To attain the status of Borland Authorized Client/Server Education Center, an organization must meet the following criteria:

- Be a member in good standing of one of the following Borland Premier Partner programs:
  * Borland Premier System Integrator
  * Borland Value-added Resellers
  * Borland Premier Alliance
  * Borland Connections for Delphi Client/Server
  * Borland Training Connections for Delphi Client/Server
- Employ at least one Borland Certified Delphi Client/Server trainer as a full-time staff member
- Sign an Authorized Client/Server Education Center Agreement or an amendment to an existing Borland partner contract
- Maintain a "current" status on Borland accounts receivable

- Maintain a formal classroom with a minimum of 8 student PC workstations and one instructor workstation meeting the following specifications:

Student Configuration:	Instructor Configuration:
486 PC or faster	486 PC or faster
16MB RAM	16MB RAM
200 MB hard disk	200 MB hard disk
VGA Monitor & card	VGA Monitor & card
MS compatible mouse	MS compatible mouse
	CD-ROM drive
	9600 Baud or faster modem

Authorized Client/Server Education Center Classroom specifications as of 5/1/96:

- Air conditioning and heating
- PC projection equipment (overhead and LCD minimum requirements)
- Projection screen
- Ability to darken room for display and presentation purposes
- 10 sq. ft. erasable writing board (white board or equivalent)
- Minimum seating for 8 hands-on student participants

Delphi Client/Server 2.0 Courseware

NUMBER : 3053
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 12, 1996

TITLE : Delphi Client/Server 2.0 Courseware

Borland, International

Delphi Client/Server 2.0 Courseware

Courseware Philosophy

Borland has developed course materials for Delphi Client/Server 2.0 Suite. The courseware is designed to emphasize concepts of Delphi Client/Server 2.0, not just techniques of using the product. There are highly structured examples integrated into the courseware which focus on key Delphi aspects central to most development. An instructional slide show, which supplements the instructor's teaching, is closely correlated to the courseware and keeps both the instructor and learner on track.

The student exercises are thought-provoking and realistic. There are additional advanced exercises for high-end students, as well as appendices for supplemental topics. The courseware becomes useful reference material after the class because of its technical depth and educational quality. This document provides a thumbnail sketch of the courseware and the environment and audience for which it is intended.

Course Description

The course is designed to ensure that Delphi Client/Server developers have an understanding of, and can successfully implement, Delphi Client/Server in a distributed computing environment.

The courseware covers the Delphi Client/Server 2.0 fundamental concepts of Delphi application development including IDE, Pascal Language basics, debugging, event-driven programming, database tasks, the Borland Database Engine, and database tools.

The bulk of the course highlights and teaches concepts of Delphi Client/Server application development including database components, working with SQL, migration to Client/Server, exception handling, component creation, and InstallShield.

Delphi Client/Server 2.0 Courseware Chapter Headings

Intended Audience

The course is targeted at those who may be new to Delphi Client/Server 2.0 Suite, but who have experience with a visual development tool.

Length of Course

The courseware is five days of planned curriculum in an instructor-led setting.

Courseware Contents

Student materials include the following:
- 150-page Courseware Manual
- 100-page Student Guide
- Exercise Diskettes

The Student Guide was created to display the slide show and printed exercises, and is a perfect place to take copious notes.

The 350-page Instructor Guide is an educationally sound combination of reference material, speaking points, sample code, and common questions.

Course Prerequisites

A working knowledge of Windows is imperative. Experience programming applications for Local Area Networks and a working knowledge of another Windows 4th generation database programming platforms (e.g. Paradox for Windows, dBASE for Windows, Visual Basic, PowerBuilder) is recommended. We encourage learners to also have experience developing applications for one of the following SQL datavase servers: InterBase 3.3/4.0, Microsoft SQL Server, Sybase, Oracle, Informix, or an ODBC supported server.

Preparation for Certification Testing

Attending a training class where this courseware is used will be one of many ways for candidates to prepare for successful completion of the Delphi Client/Server certification exam. However, taking this course and learning the material contained within may not be sufficient preparation for passing the test. The courseware covers a broad set of features and concepts many of which are included on the test. The training materials were not written as the sole preparation for certification, but instead to educate and prepare people for a real-world Delphi development environment.

Courseware Availability

The official Borland Delphi Client/Server 2.0 courseware is taught by Delphi Client/Server Certified Training Professionals only. These professionals have demonstrated a high level of technical skills and competence through Delphi Client/Server certification and completed a rigorous train-the-trainer program for this courseware.

Pricing (U.S. and Canada)

Only Certified Training Professionals can order these materials.

	Certified Training Professional	Authorized Education Center
Student Manual	$275/each	$225/each
Instructor Guide	$500/each*	$500/each*

* One Instructor Guide is included in each Train-the-Trainer

course. You must be a certified Delphi Client/Server trainer to order a replacement guide.

How to Order

There are three ways to order courseware from Crestec Los Angeles, Inc., Borland's printer and distributor of Delphi training materials. You can use fax, email or the automated attendant (voicemail) 24 hours a day, seven days a week. Monday through Friday during business hours, you will receive a confirmation within three hours of your order. Order forms are available on Borland's web site and enclosed with each order.

Fax Number: 310-715-9761
Email Address: borland@crestecla.com
Toll-free Voicemail Number: 888-234-4553

Study Objectives for the Delphi Client/Server Exam

NUMBER : 3054
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 12, 1996

TITLE : Study Objectives for the Delphi Client/Server Exam

Borland International

Study Objectives
for the Delphi Client/Server Exam

Borland's Delphi Client/Server Exam is a comprehensive test of knowledge and skills of the Delphi Client/Server technology. This document is for individuals who are seeking Delphi Client/Server certification and are preparing to take the Exam. The information provided is a list of knowledge objectives that can be used as a study guide for the Delphi Client/Server Exam. It is intended to help you understand the content of the Exam so you can effectively study and achieve your certification goals. It is also recommended that you take an official Borland Delphi Client/Server Training Course and use Borland Press Books to prepare.

The Delphi Client/Server Exam will be administered by Sylvan Prometric at over 200 testing centers worldwide. A limited number of discounted test seats will also be offered on site at Borland's Developer Conference (BDC) in Anaheim, CA, July 29 & 30, 1996. To register for Delphi Client/Server Certification at BDC or at a Sylvan Prometric testing center near you, call Sylvan Prometric 1-800-430-EXAM (1-800-430-3926). The test fee for the United States and Canada is $140. For International test fees, contact your local Borland sales office.

Windows

- Describe the features of the Windows API relevant to Delphi.
- Describe the correct use of the TOLEContainer to embed an object.
- Describe characteristics of OLE and DDE. Given a situation identify the correct use of each.

Object Pascal

- Define the major sections of a unit (i.e. interface; uses; implementation; initialization).
- Describe procedural and variable scope.
- Describe when to use virtual and override and given a situation, describe the correct use of each.
- Identify the differences between constructors and destructors and, given a situation, identify the correct use of each.
- Describe the methods available for destroying objects.

- Identify and define the major sections of a class declaration (i.e., class(...), private, protected; public; published).
- Describe when to use try...finally and try...except, and given a situation, identify the correct usage of each.
- Describe when it is necessary to use raise and given a situation, identify its correct usage.
- Identify the methods which access object-type information at run-time (e.g., is, ClassType; as; etc.). Given a situation identify the correct use of each.

IDE

- Identify the files (e.g. .dpr; .dfm; .pas; etc.) that make up a project.
- List the steps required to make a DLL project. Identify a properly constructed DLL project.
- Identify the steps to create and destroy forms dynamically. Given a situation identify the correct use of the Create and Free methods.
- Describe when it is appropriate to use Show vs. ShowModal (i.e. overlapped windows vs. dialogs). Given a situation identify the correct use of each.
- List the Project and Environment options which are useful for development vs. deployment (e.g. "Range Checking", "Break on Exception"). Given a situation, identify the correct usage.

Components

- Describe the differences between ownership (e.g. Owner, Components[], etc.) and parenthood (e.g. Parent, Controls[], etc.) with respect to a child.
- Identify the properties and methods available for creating a drag-and-drop application. Given a situation identify their correct usage (e.g., DragMode, BeginMode, etc.).
- Define the role of the ModalResult property in relation to the closing of a form.
- Given a situation identify the correct use of the GroupIndex property as it applies to several related components (i.e., TSpeedButton, TMenuItem).
- Given a situation identify the correct use of the Cursor property (e.g., set to an hourglass during repetitive process).
- List some of the properties and methods related to TCanvas (e.g., Brush, FillRect,, etc.). Given a situation identify their correct usage.

Database Components (1)

- Identify the methods which affect a TDataSet.
- Identify the issues involved in creating a calculated field.
- Describe how to search for a record using keyed fields vs. non-keyed fields.
- Describe the functionality of the TQuery's DataSource property. Describe how it is related to the TTable's MasterSource property.
- Identify the types of SQL which are used with the TQuery components.

- Identify the issues involved when using the Prepare method of the TQuery component?
- Given a situation, tell how to correctly assign values to a TField.
- Demonstrate knowledge of the Session global variable, including its function, scope, and where it is created.

- List the properties and methods available for manipulating fields in a TDBGrid. Given a situation identify the correct usage.

Database Tasks (1)

- Identify properties and methods used to traverse a dataset.
- Identify the correct use of the TDataSet events.
- Identify the properties and methods used to format and validate database fields at runtime.

Component Design

- Identify the steps required to create and register a custom component.
- Identify the steps required to create a property and an event for a component.

SQL

- Given a situation identify the differences between pass-through SQL and standard.
- Given a situation identify the correct result of the SELECT statement.
- Given a situation identify the correct use of the DELETE statement.
- Given a situation identify the correct use of the INSERT statement.
- Given a situation identify the correct use of the UPDATE statement.
- Identify the correct use of the CREATE TABLE statement.
- Identify the correct use of the CREATE VIEW statement.
- Given a situation, identify the correct use of a CREATE INDEX statement.

Database Components (2)

- List the components, properties and methods necessary for server interaction (i.e., TDatabase). Given a situation identify their correct usage (e.g., logging-on to a server without prompting the user).
- Identify the necessary steps in creating your own login dialog.
- Demonstrate knowledge of the AliasName and the DriverName properties of the TDatabase component.
- Identify the correct usage of StartTransaction, Commit, Rollback and their related behavior.
- List the advantages of using a TDatabase component.
- Describe the purpose of TDatabase's KeepConnection property.

- Demonstrate knowledge of TDatabase's UpdateMode property.
- Identify the record searching methods of a dataset.
- Demonstrate familiarity with the properties and methods of the TStoredProc component.
- Demonstrate competence using Params and ParamByName to satisfy the parameters of dynamic queries.
- Demonstrate knowledge of moving data using TBatchMove and other Delphi components.
- Describe the advantages and disadvantages which arise when using TTable components vs. TQuery and TStoredProc components on remote databases (i.e., network traffic).

Database Tasks (2)

- Describe how to deal with referential integrity constraints using triggers and stored procedures.
- Given a situation identify the correct use of SQL exceptions (e.g., warning of a key violation).
- Given a situation identify the correct use of grant.
- Describe the locking mechanisms used by the BDE (e.g., optimistic locking).

Up-Sizing Applications

- Given a situation identify the correct uses of the TBatchMove component.

Borland Database Engine

- Identify the available settings in the BDE configuration for configuring a SQL connection.
- Describe how to manually create and edit a BDE alias.

ReportSmith

- Identify the properties of the TReport component.
- Identify the steps required to avoid the connect dialog when loading a report.

Delphi Client/Server 2.0 Train-the-Trainer Class

NUMBER : 3055
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : June 12, 1996

TITLE : Delphi Client/Server 2.0 Train-the-Trainer Class

Borland, International

Delphi Client/Server 2.0 Train-the-Trainer

The Train-the-Trainer class for Delphi Client/Server 2.0 is an essential requirement for technical trainers who want to boost their competency in the classroom and complete the Delphi Client/Server Certification Program.

The class incorporates training strategies to showcase Delphi Client/Server 2.0 courseware and is taught by technical and educational specialists who have been personally involved in the instructional design of the Delphi Client/Server 2.0 courseware.

As the cornerstone for trainer certification, this class provides an opportunity for technical trainers to build on their already sound product knowledge, expand their courseware knowledge, and hone their classroom presence. This is a dynamic, fast-paced program which emphasizes practice and participation in order to build skills and confidence of our valued group of Delphi Client/Server training professionals.

Objectives of Delphi Client/Server 2.0 Train-the-Trainer

- Provide a recognized process for trainers seeking Delphi Client/Server 2.0 trainer certification.
- Ensure the high quality of instructors representing and teaching Delphi Client/Server 2.0.
- Build consistency among the instructors using the official Borland courseware.

General Overview

Daily section reviews, sample exercises, and Train-the-Trainer discussions provide the structure for the class and allow individuals to systematically progress through the material. Student teachings are also a major component of the class. Each of the instructors in attendance has a unique style, teaching methods and Delphi knowledge from which the class can benefit. Therefore, we want you in the instructor role part of the time. Both individual and team teaching is available. The teaching modules are integrated with the instructor's daily agenda.

Who Should Attend

This class is open to those instructors who are working toward completion of the Delphi Client/Server Trainer Certification Program. All of the following prerequisites are recommended:
- at least two years of technical training experience
- successfully passing the Delphi Client/Server Exam
- attendance at a Delphi class using the official Borland Delphi Client/Server 2.0 courseware (this may be waived if you have been teaching Delphi 1.0 or successfully pass the Exam in two attempts)
- evidence of superior presentation skills

Cost

$2995/student

This price includes one Delphi Client/Server Instruction Guide, ten copies of the Client/Server 2.0 courseware for your first class, and five free courseware support phone calls to Borland's courseware developer, all valued at over $2800.

Length

4.5 days

Class Size
---
12 students

How to Register

For specific dates, locations, or to request a registration form for upcoming Train-the-Trainer classes, please call Lisa Coenen at 408-431-5758.

Cancellation Policy

If a student cancels a class within 10 business days prior to the start date of the course, a full refund may be obtained. Any cancellation or rescheduling with less than 10 business days will cause the student to incur a 50% late cancellation penalty. Registrants who do not cancel and do not attend will forfeit the entire course fee. All cancellations must be in writing.

Course Cancellation

Borland reserves the right to cancel any training class. Should Borland cancel a class, a full refund of tuition or fees will be issued to the student or applied as credit toward a rescheduled class. Borland cannot assume responsibility for any other costs to the student (i.e. non-refundable airline tickets).

you received with the Borland product to which this information pertains.

Redistributing Applications using the ISP

NUMBER : 3078
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : December 13, 1996

TITLE : Redistributing Applications using the ISP

Redistributing Applications using the Internet Solutions Pack
---

Applications built with the Internet Solutions Pack can be redistributed using most third party installation programs, including the InstallShield Express utility that ships with Delphi 2.0 Client/Server and Developer. In order to deploy your application, any OCX controls you used in building the application need to be redistributed along with your executable, as well as any other necessary files (eg. dll's) your program requires.

In addition, the Internet controls are dependent upon NetManage's support dll's, and must be included with your program. The following list contains of all the OCX files and NetManage DLLs that ship with the Internet Solutions Pack (and that may need to be redistributed):

OCX files

HTML.OCX
HTTPCT.OCX
NNTPCT.OCX
POPCT.OCX
SMTPCT.OCX
WINSCK.OCX
FTPCT.OCX

NetManage DLLs

NMOCOD.DLL
NMSCKN.DLL
NMFTPSN.DLL
NMW3VWN.DLL
NMORENU.DLL

For a list of which files you actually need to include with your distribution, use the following dependency matrix:

	Winsock	FTP	NNTP	SMTP	POP3	HTTP	HTML
NMSCKN.DLL	X	X	X	X	X	X	X

NMORENU.DLL		X	X	X	X	X	X
NMOCOD.DLL		X	X	X	X	X	X
WINSCK.OCX	X						
FTPCT.OCX		X					
NMFTPSN.DLL		X					
NNTPCT.OCX			X				
SMTPCT.OCX				X			
POPCT.OCX					X		
HTTPCT.OCX						X	
HTML.OCX							X
NMW3VWN.DLL							X

Patches:

The following patches are available for download from the Borland Delphi 32 CompuServe Forum (BDELPHI32) and the Borland website (www.borland.com).

NMPATCH.ZIP - Unzip this file and follow the included directions. This patch was supplied by NetManage and will actually update the controls themselves.

ISP.DCU - Place in the Delphi 2.0\lib subdirectory and recompile your project. This will prevent the error message about not being licensed.

To deploy your program using InstallShield Express, follow these steps:

1) Start up ISXpress, and build your Setup project as you normally would (if you are unfamiliar with building a Setup project with InstallShield Express, refer to the InstallShield online help).

2) In the Components & Groups portion of the setup, add your executable file into an appropriate group (eg. the Program Files group (default)).

3) Create a new group for the shared files (eg. "Shared Files"), and set the Destination Directory to <WINSYSDIR>. Then launch the Explorer and drag the necessary files (as determined by using the above matrix) to the newly created folder. These files should be located in your Windows\System directory on your hard drive (if not, you can find them on the Delphi 2.0

CD-ROM under \RUNIMAGE\DELPHI20\WINDOWS\SYSTEM32).

4) Finish the rest of the Setup procedure.

You are now ready to deploy your application. InstallShield will automatically register all OCX's and DLLs that have <WINSYSDIR> set as their destination directory.

Sharing Violation Error with Paradox Tables

NUMBER : 3089
PRODUCT : BDE
VERSION : All
OS : Windows
DATE : August 14, 1996

TITLE : Sharing Violation Error with Paradox Tables

Description:
This error is most commonly caused by a "Lock File Contention". A lock file contention occurs in this situation:
One user, User "A", is accessing one or more Paradox tables in a directory. "A" is closing the last table in the directory. The BDE detects that "A" is the last program accessing the tables in the directory. Since "A" is the last user on the tables, the .LCK files are going to be deleted. At this time, User "B" attempts to open a table in the same directory as "A" (who is about to delete the .LCK files). "B" opens the .LCK files to write an entry to. This is when "A" tries to delete the .LCK files that "B" has now opened. This will cause a sharing violation on the .LCK files. The share violation usually occurs on the last client to close the table.

Solutions:
One solution is to override Windows error routines and ignore the error. In all test cases, then a sharing violation occurs, the table is still opened or closed.
Another solution is to keep the .LCK files in the table's directories. If the BDE detects that more than one Application (or Session) is using the tables, it will not attempt to delete the .LCK files therefore solving the problem.

1) C / C++ / Delphi / Paradox:
Use the Windows API SetErrorMode passing the SEM_FAILCRITICALERRORS constant. You would only need to do this before opening and closing a table and then check to see if the table has actually been open. Please see Routine 1a and 1b below for a basic Delphi (any version) routine or Routine 2a and 2b for C / C++ routines. Paradox users will need to call into the appropriate DLL where SetErrorMode resides.

2) C / C++ / Delphi / Paradox:
Create a "Dummy" Paradox table in each table directory that the application uses. Example: executable is in directory C:\MyProg. This executable opens tables in two different directories - C:\MyTables

and D:\TempTbl. Create a "Dummy" table with the Database Desktop, or any other Paradox table creation utility, in both the C:\MyTables and D:\TempTbl directories. Once the tables are created, create an application that opens each of the "Dummy" programs at startup. Leave this program running at all times. If the executable can be placed on a server machine, like Windows NT, that is optimal. In this scenario, whenever applications are accessing the data, the server also has the "Dummy" tables open.

3) C / C++ / Delphi / Paradox:
Leave at least one table during the entire application's run. Open one table during startup and close the table at exit. You could even open a "Dummy" table described above.

4) C / C++ / Delphi:
Use DbiAcqPersistTableLock BDE function to create a lock on a table that does not exist (Delphi or C++ only).

Syntax:
(Delphi) Check(DbiAcqPersistTableLock(Database1.Handle, 'NoTable1.DB', szPARADOX));

(C++) rslt = DbiAcqPersistTableLock(hDb, 'NoTable2.DB', szPARADOX);

NOTE: Each instance of the application must have a unique, non-existent table name or an attempt to place a lock in the directory will fail. If this method is used, an algorithm must be used to guarantee a unique table name. If a user can only run one instance of the program at a time, the network user name can be used as a table name.

5) Paradox ObjectPAL:
If your application is apt to cause .LCK files to be created and deleted often, the easiest (and least resource-intensive) way to prevent this is to place a read lock on a non-existent table within the directory when your application starts up. (Paradox allows us to place semaphore locks on tables that don't exist.) Since read locks don't conflict with one another, all users can do this, and the net result will be that the .LCK file will not be deleted until the last user exits the system.

Syntax:
At program startup:
Table.attach("DummyTbl.db")
Table.lock(Read)

At program shut-down:
Table.unlock(Read)
Table.unAttach

NOTES - These are observations on the problem:

1) The problem does not seem to occur on NT workstation machines. The application can be either 16 or 32 bit. SetErrorMode seems to work with both cases.

2) The problem is more likely to occur if the network protocol is netBEUI. If possible use netBIOS or IPX/SPX.

3) Constant closing and opening of tables will cause this error more often.

4) The error most commonly occurs on the close of the table, not the open. In this situation the closing application tries to delete files that another application now has open.

Example Routines:

Routine 1a) (Delphi Table Open Routine)

```
procedure TForm1.OpenTable(MyTable: TTable);
begin
  { Set the error mode to ignore critical errors }
  SetErrorMode(SEM_FAILCRITICALERRORS);
  { Open the table and check if it was opened }
  MyTable.Open;
  if MyTable.Active = False then
  begin
    { Retry opening the table }
    MyTable.Open;
    { If an error happens again, raise an exception }
    if MyTable.Active = False then
      raise
        EDatabaseError.Create('Error Opening table');
  end;
  { Set the error mode back to the default }
  SetErrorMode(0);
end;
```

Routine 1b) (Delphi Table Close Routine)

```
procedure TForm1.CloseTable(MyTable: TTable);
begin
  { Set the error mode to ignore critical errors }
  SetErrorMode(SEM_FAILCRITICALERRORS);
  { Close the table and check if it was closed }
  MyTable.Close;
  if MyTable.Active = True then
  begin
    { Retry closing the table }
    MyTable.Close;
    { If an error happens again, raise an exception }
    if MyTable.Active = True then
      raise
        EDatabaseError.Create('Error Closing table');
  end;
  { Set the error mode back to the default }
  SetErrorMode(0);
end;
```

Routine 2a) (C / C++ Table Open Routine)

```
---------------------------------------
DbiResult OpenTable(hDBIDb hTmpDb, pCHAR szTblName,
                    phDBICur phTmpCur)
{
    DBIResult    rslt;

    // Set the error mode to ignore critical errors
    SetErrorMode(SEM_FAILCRITICALERRORS);

    // Open the table and check if it was opened
    rslt = DbiOpenTable(hTmpDb, szTblName, NULL,
        NULL, NULL, NULL, dbiREADWRITE, dbiOPENSHARED,
        xltFIELD, FALSE, NULL, phTmpCur);
    if (rslt != DBIERR_NONE)

        // Retry opening the table
        rslt = DbiOpenTable(hTmpDb, szTblName, NULL,
            NULL, NULL, NULL, dbiREADWRITE, dbiOPENSHARED,
            xltFIELD, FALSE, NULL, phTmpCur);

    // Set the error mode back to the default
    SetErrorMode(0);
    return rslt;
}
```

Routine 2b) (C / C++ Table Open Routine)

```
---------------------------------------
DBIResult CloseTable(phDBICur phTmpCur)
{
    DBIResult    rslt;

    // Set the error mode to ignore critical errors
    SetErrorMode(SEM_FAILCRITICALERRORS);
```

```
    // Close the table and check if it was closed
    rslt = DbiCloseCursor(phTmpCur);
    if (rslt != DBIERR_NONE)

        // Retry closing the table
        rslt = DbiCloseCursor(phTmpCur);

    // Set the error mode back to the default
    SetErrorMode(0);
    return rslt;
}
```

How to Create a TDBGrid Lookup Field in Delphi 2.0

NUMBER : 3096
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 21, 1996

TITLE : How to Create a TDBGrid Lookup Field in Delphi 2.0

How to create a lookup field in a TDBGrid for Delphi 2.0

1. Drop 2-TTable's, 1-TDataSource and 1-TDBGrid on a form.

2. Connect Table1 to DataSource1 to DBGrid1
   a. DataSource1.DataSet = Table1
   b. DBGrid1.DataSource = DataSource1

3. Setup Table1
   a. Table1.Database = DBDemos
   b. Table1.TableName = Customer
   c. Table1.Active = True

4. Setup Table2
   a. Table2.Database = DBDemos
   b. Table2.TableName = Orders
   c. Table2.Active = True

5. Add all of the fields for Table1 by bringing up the Fields Editor:
   a. Double click on Table1
   b. Right click on Fields Editor
   c. Add New Fields. Add all of them

6. Add a new field for Table1.
   a. Right click on Fields Editor, and select New Field.

7. Specify the following parameters for the newly added field.
   a. Name: Bob
   b. Type: String
   c. Size: 30
   d. Select Lookup
   e. Key Fields: CustNo - Field in Table1 to store value
   f. DataSet: Table2 - Table lookup is being done on
   g: LookUpKeys: CustNo - This Key gets copied to KeyField
   h: Result Field: OrderNo - Value to display to the user in the drop down box

8. Run the application

Dynamically Creating Page Controls and Tab Sheets

NUMBER : 3097
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 20, 1996

TITLE : Dynamically Creating Page Controls and Tab Sheets

```
{ Create a Page Control and all the TabSheets and buttons dynamically }
{ Delphi 2.0 or greater }

var
  T : TTabSheet;
  P : TPageControl;
begin
   // Create the PageControl
   // need to reference the page control so we need a reference to it.
  P := TPageControl.Create(application);
  with P do begin
    Parent := Form1;   // set how controls it.
    Top := 30;
    Left := 30;
    Width := 200;
    Height := 150;
    end;   // with TPageControl

    // Create 3 pages
  T := TTabSheet.Create(P);
  with T do begin
    Visible := True;     // This is necessary or form does not repaint
                         // correctly
    Caption := 'Page 1';
    PageControl := P;   // Assign Tab to Page Control
    end;   // with

  T := TTabSheet.Create(P);
  with T do begin
    Visible := True;     // This is necessary or form does not repaint
                         // correctly
    Caption := 'Page 2';
    PageControl := P;   // Assign Tab to Page Control
    end;   // with

  T := TTabSheet.Create(P);
  with T do begin
    Visible := True;     // This is necessary or form does not repaint
                         // correctly
    Caption := 'Page 3';
    PageControl := P;   // Assign Tab to Page Control
    end;   // with

    // Create 3 buttons, 1 per page
  with tbutton.create(application) do begin
```

```
    Parent := P.Pages[0];   // Tell which page owns the Button
    Caption := 'Hello Page 1';
    Left := 0;
    Top := 0;
    end; // with

  with tbutton.create(application) do begin
    Parent := P.Pages[1];   // Tell which page owns the Button
    Caption := 'Hello Page 2';
    Left := 50;
    Top := 50;
    end; // with

  with tbutton.create(application) do begin
    Parent := P.Pages[2];   // Tell which page owns the Button
    Caption := 'Hello Page 3';
    Left := 100;
    Top :=   90;
    end; // with

    // This needs to be done or the Tab does not sync to the
    // correct page, initially.   Only if you have more then
    // one page.
  P.ActivePage := P.Pages[1];
  P.ActivePage := P.Pages[0];   // page to really show
end;
```

Navigating a Multiselected Listbox

NUMBER : 3098
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : August 20, 1996

TITLE : Navigating a Multiselected Listbox

Navigating a multiselected listbox

This example shows a message for every element in a listbox that has been selected by the user.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Loop: Integer;
begin
  for Loop := 0 to Listbox1.Items.Count - 1 do begin
    if Listbox1.Selected[Loop] then
      ShowMessage(Listbox1.Items.Strings[Loop]);
  end;
end;
```

Making Your Delphi 2.0 Applications "Sing"

NUMBER : 3099
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 20, 1996

TITLE : Making Your Delphi 2.0 Applications "Sing"

```
(*
This TI demonstrates how to make your Delphi 2.0
application "sing" by loading and playing a wave file
four different ways:

1) Use the sndPlaySound() function to directly
play a wave file.

2) Read the wave file into memory, then use the
sndPlaySound() to play the wave file

3) Use sndPlaySound to directly play a wave
file thats embedded in a resource file attached
to your application.

4) Read a wave file thats embedded in a resource
 file attached to your application into memory,
 then use the sndPlaySound() to play the wave file.

 To build the project you will need to:

1) Create a wave file called 'hello.wav'
in the project's directory.

2) Create a text file called 'snddata.rc'
in the project's directory.

3) Add the following line to the file 'snddata.rc':
HELLO WAVE hello.wav

4) At a dos prompt, go to your project directory
and compile the .rc file using the Borland Resource
compiler (brcc32.exe) by typing the path to brcc32.exe
and giving 'snddata.rc' as a parameter.

Example:

bin\brcc32 snddata.rc

This will create the file 'snddata.res' that
Delphi will link with your application's .exe
file.

Final Note: Keep on Jamm'n!
```

```
*)

unit PlaySnd1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    PlaySndFromFile: TButton;
    PlaySndFromMemory: TButton;
    PlaySndbyLoadRes: TButton;
    PlaySndFromRes: TButton;
    procedure PlaySndFromFileClick(Sender: TObject);
    procedure PlaySndFromMemoryClick(Sender: TObject);
    procedure PlaySndFromResClick(Sender: TObject);
    procedure PlaySndbyLoadResClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{$R snddata.res}

uses MMSystem;

procedure TForm1.PlaySndFromFileClick(Sender: TObject);
begin
  sndPlaySound('hello.wav',
                    SND_FILENAME or SND_SYNC);
end;

procedure TForm1.PlaySndFromMemoryClick(Sender: TObject);
var
  f: file;
  p: pointer;
  fs: integer;
begin
  AssignFile(f, 'hello.wav');
  Reset(f,1);
  fs := FileSize(f);
  GetMem(p, fs);
  BlockRead(f, p^, fs);
  CloseFile(f);
```

```
  sndPlaySound(p,
                  SND_MEMORY or SND_SYNC);
  FreeMem(p, fs);
end;

procedure TForm1.PlaySndFromResClick(Sender: TObject);
begin
  PlaySound('HELLO',
            hInstance,
            SND_RESOURCE or SND_SYNC);
end;

procedure TForm1.PlaySndbyLoadResClick(Sender: TObject);
var
  h: THandle;
  p: pointer;
begin
  h := FindResource(hInstance,
                    'HELLO',
                    'WAVE');
  h := LoadResource(hInstance, h);
  p := LockResource(h);
  sndPlaySound(p,
                  SND_MEMORY or SND_SYNC);
  UnLockResource(h);
  FreeResource(h);
end;


end.
```

How to check a ComboBox without OnClick ocurring.

NUMBER : 3009
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : October 23, 1996

TITLE : How to check a ComboBox without OnClick ocurring.

When you modify the state of a TCheckBox by setting the value of its Checked property, the OnClick event of that TCheckBox is fired. For example, the following code:

```
CheckBox1.Checked := True;
```

causes CheckBox1.OnClick to execute.

Sometimes, however, you may want to check or uncheck the CheckBox in code without firing the OnClick event. You can do this by sending a BM_SETCHECK message to the ComboBox. The WParam of this message can be 1 (meaning the box should be checked), or 0 (meaning the box should be unchecked). The LParam of this message is always 0.

With this in mind, the following procedure accepts as parameters a TCheckBox called CB and a Boolean parameter called CheckIt. When CheckIt is True, the box will be checked, and when it's False, the box will be unchecked:

```
procedure CheckNoClick(CB: TCheckBox; CheckIt: Boolean);
begin
  CB.Perform(BM_SETCHECK, Ord(CheckIt), 0);
end;
```

Using this procedure, the code for checking a TComboBox called ComboBox1 looks like this:

```
CheckNoClick(ComboBox1, True);
```

Making Accelerators Work with a TPageControl

NUMBER : 3101
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 21, 1996

TITLE : Making Accelerators Work with a TPageControl

The TPageControl found on the Win95 page of the Component Palette does not currently work with accelerators. However, it is possible to create a descendant of TPageControl which includes this feature.

The component shown in the code to follow is such a control. This TPageControl descendant captures the CM_DIALOGCHAR message. This allows you to capture key strokes that may be accelerators for a given form. The CMDialogChar message handler uses the IsAccel function to determine if the captured key code refers to an accelerator on one of the TPageControls pages. If so, the appropriate page is made active and is given focus.

```
unit tapage;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls;

type

  TAPageControl = class(TPageControl)
  private
    procedure CMDialogChar(var Msg: TCMDialogChar); message
     CM_DIALOGCHAR;
  end;

procedure Register;

implementation

procedure TAPageControl.CMDialogChar(var Msg: TCMDialogChar);
var
  i: Integer;
  S: String;
begin
  if Enabled then
    for I := 0 to PageCount - 1 do
      if IsAccel(Msg.CharCode, Pages[i].Caption) and
        Pages[I].TabVisible then begin
        Msg.Result := 1;
        ActivePage := Pages[I];
        Change;
```

```
            Exit; // exit for loop.
          end;
    inherited;
end;

procedure Register;
begin
    RegisterComponents('Test', [TAPageControl]);
end;

end.
```

How to Dynamically Create A Page Control

NUMBER : 3102
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 21, 1996

TITLE : How to Dynamically Create A Page Control

This document demonstrates how to dynamically add tab sheets (TTabSheet objects) to a Windows 95/NT Page Control (TPageControl object). Both of these objects are declared in the ComCtrls unit. So make sure that ComCtrls is listed in the 'uses' clause on your form.

HOW TO DYNAMICALLY CREATE A PAGE CONTROL

Before we get into dynamically creating tab sheets, lets first discuss how to dynamically create a PageControl (if one isn't on the form already). This is done by calling TPageControl's Create constructor with an owner parameter of Self. The Create constructor returns a object reference of the newly created Page Control object and assigns it to the 'PageControl' variable. The second step is to set PageControl's Parent property to Self. The Parent property determines where the new PageControl is to be displayed; in this case its the form itself. Here's a code snippet that demonstrates this.

```
var
  PageControl : TPageControl;

PageControl := TPageControl.Create(Self);
PageControl.Parent := Self;
```

Note: When the form gets destroyed the Page Control and it tab sheets will be destroyed also because they are owned by the form.

HOW TO DYNAMICALLY CREATE A TAB SHEET

There are two basic steps to dynamically add a new page to a PageControl. The first is to dynamically create the TTabSheet as follows:

```
var
  TabSheet : TTabSheet;

TabSheet := TTabSheet.Create(Self);
```

Then we need to give it a caption as follows:

```
TabSheet.Caption := 'Tabsheet 1';
```

And finally, the most important piece is to tell the new tab sheet which Page Control it belongs to. This is done by assigning the TTabSheet's PageControl property a TPageControl reference variable like the one created above (PageControl). Here's a code snippet that demonstrates this.

```
TabSheet.PageControl := PageControl;
```

HOW TO DYNAMICALLY ADD A CONTROL TO A TAB SHEET

The key to creating and placing any control on a tab sheet is to assign it's Parent property a TTabSheet class reference variable. Here is an example.

```
var
  Button : TButton;

Button := TButton.Create(Self);
Button.Caption := 'Button 1';
Button.Parent := TabSheet;
```

For more information on the TPageControl and TTabSheet objects refer to the on-line documentation as well as look at the ComCtrls.pas file in your ..\Delphi 2.0\SOURCE\VCL directory.

FULL SOURCE EXAMPLE

```
// This code is extracted from a form with a single button on it.

unit DynamicTabSheetsUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure TestMethod(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

uses ComCtrls;

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  PageControl : TPageControl;
  TabSheet : TTabSheet;
begin
  // Create the PageControl
  PageControl := TPageControl.Create(Self);
  PageControl.Parent := Self;

  // Create 1st page and associate it with the PageControl
  TabSheet := TTabSheet.Create(Self);
  TabSheet.Caption := 'Tabsheet 1';
  TabSheet.PageControl := PageControl;

  // Create the first page

  with TButton.Create(Self) do
  begin
    Caption := 'Button 1';
    OnClick := TestMethod;   // Assign an event handle
    Parent := TabSheet;
  end;

  // Create 2nd page and associate it with the PageControl

  TabSheet := TTabSheet.Create(Self);
  TabSheet.Caption := ' Tabsheet 2';
  TabSheet.PageControl := PageControl;
end;

procedure TForm1.TestMethod(Sender: TObject);
begin
  ShowMessage('Hello');
end;

end.
```

Connecting to a 32-bit Sybase server

NUMBER : 3152
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Connecting to a 32-bit Sybase server

Connecting To Sybase 32-bit

This TI will instruct you on how to connect to a Sybase database via Borland's native 32-bit Sybase SQL Links packaged with Delphi 2.x. The Sybase client software should take up approximately 10+ megabytes of hard drive space.

Steps To Connect:
1. Make sure that the SQL Links package is installed on your local drive. If you installed the full package of Delphi 2.x then it should be installed already.
2. Install the Sybase client software.
3. When prompted during installation to install either the 16 or 32 bit Sybase links, choose only 32-bit by checking the check box next to the 32-bit selection and making sure that the check box next to the 16-bit selection is blank.
4. After the client software is installed on your hard drive, you will be asked if you would like the install program to automatically update your AUTOEXEC.BAT file. You should choose YES.
5. When prompted to edit your SQL.INI file, choose yes.
6. In the "Input Server Name:" section, type in the alias of the server. Click on the 'Add' button to add this name to your "Server Entry:" list. Next, make sure that the "Service Type:" (It should be 'query'), the "Platform:" (it usually defaults to either NT, dos, or Win3), and the "Net-Library Driver:" (should default to either NLWNSCK or NLNWLINK) edit boxes are correct. Fill out the "Connection Information/Network Address:" by entering the network address of the server you will be connecting to in this edit box. click on the 'Add Service' button. You should now be able to Ping your server by clicking on the 'Ping' button. Exit and save the current settings.
7. Shutdown and Restart your machine.
8. Goto the Delphi program group and execute the Database Explorer.
9. In the Database explorer, make sure that you are on the Database tab. In the pull down menus, choose Object | New... A dialog box should appear with the name STANDARD next to a down arrow. Click the down arrow and select SYBASE from the list that is displayed.
10. There should now be an Alias for your Sybase connection called SYBASE1. Make sure that this name is highlighted. Click on the definitions tab in the Database Explorer. Under the "Server Name" section, type the name of one of the servers that you entered in your SQL.INI that you were able to Ping. In the "User Name" section type the name of a user with rights to the server specified in your "Server Name" section. Make sure that you know the password of the user name that you just specified.

11. Double click on the alias name (SYBASE1) and you should be prompted for a User Name and Password. The User Name should default to the name you specified in the "User Name" section of the Sybase Alias. Type in the password that corresponds to this User. Click on the OK button. You should now see a little green box around the icon next to the Sybase Alias (SYBASE1). This means that you are connected.

Testing your Connection within Delphi 2.x:

1. Drop a TDataSource, a TTable, and a TDBGrid on a blank form.
2. In the Object Inspector for the TDataSource, Set the DataSet to 'Table1'(no quotes).
3. In the Object Inspector for the TTable, set the DataBase Name to SYBASE1. Scroll down to the TableName property and double click on the edit box next to it. You should be prompted for a user name and a password. The user name that you entered in the Database Explorer for the Sybase Alias should already be displayed. Enter the corresponding password. Click the OK button.
4. You should now see a list of table names. Choose one of these names.
5. Click on the TDBGrid. Set the DataSource Property to DataSource1.
6. Set the Active property of the TTable to TRUE.
7. You should now see the data in the TDBGrid. When you run this application, you will be prompted for the username/password. Type in your password and click OK. You should now see the data in the grid.

Error Messages:
Unable to determine net-library error: This error means that the .DLL's needed are nowhere to be found. These files should be in your \Sybase\DLL directory:
Libblk.dll
Libcomn.dll
Libcs.dll
Libct.dll
Libintl.dll
Libsrv.dll
Libsybdb.dll
Libtcl.dll
Mscvrt10.dll
Nldecnet.dll
Nlmsnmp.dll
Nlnwadvt.exe
Nlnwlink.dll
Nlwnsck.dll

Disclaimer: This document does not promise a connection, it only shows the best and fastest way to connect.

Accessing Paradox Tables on CD or Read-Only Drive

NUMBER : 3104
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : August 20, 1996

TITLE : Accessing Paradox Tables on CD or Read-Only Drive

This Technical Information document will step through the concepts regarding accessing Paradox tables which are located on a CD-ROM or any read-only device.

The Paradox locking scheme requires the existence of a PDOXUSRS.LCK file to handle its locking logic. This file is generally created at run-time and resides in the directory which also contains the tables. However, with a CD-ROM there is not a way to create this file at run-time on the CD-ROM. The solution is simple, we create this file and put it on the CD-ROM when the CD is pressed. The following steps will give you a very simple utility program for creating the PDOXUSRS.LCK file which you will then copy to the CD-ROM image.

1. Starting with a blank project add the following components: TEdit, TButton and TDatabase.

2. In the OnClick event for the button use the following code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ChkPath then
    Check(DbiAcqPersistTableLock(Database1.Handle,
                'PARADOX.DRO','PARADOX'));
end;
```

3. The ChkPath function is a user defined method of the form. It will simply check the path entered in the Edit box and make sure it exists. Here is the function:

```
function TForm1.ChkPath : Boolean;
var
  s : array[0..100] of char;
begin
  If DirectoryExists(Edit1.Text) then begin
    DataBase1.DatabaseName:= 'TempDB';
    DataBase1.DriverName:= 'Standard';
    DataBase1.LoginPrompt:= false;
    DataBase1.Connected := False;
    DataBase1.Params.Add('Path=' + Edit1.Text);
    DataBase1.Connected := TRUE;
    Result := TRUE;
  end
  else begin
```

```
      StrPCopy(s,'Directory : ' + Edit1.text + ' Does Not Exist');
      Application.MessageBox(s, 'Error!', MB_ICONSTOP);
      Result := FALSE;
    end;
end;

{ Note: Don't forget to put the function header in the public section
          of the form.}
```

4. There is one more thing you need to add before compiling, in the Uses statement at the top of the unit add the following units:
   Delphi 1.0: FileCtrl, DbiProcs, DbiTypes, DbiErrs.
   Delphi 2.0: FileCtrl , BDE

When you have compiled and executed the utility program, it will create two files in the directory you specified. The two files created are: PDOXUSRS.LCK and PARADOX.LCK.

Note: The PARADOX.LCK file is only necessary when accessing Paradox for DOS tables so you can delete it.

5. The only thing left for you to do is copy the remaining file (PDOXUSRS.LCK) to the CD-ROM image. Of course your tables will be Read-Only.

Note: If you want to clean up this utility for future use, you can change the text property of the Edit box to be some default directory and change the Caption property of the Button to be something more meaningful.

Here is the final version of the code:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,   DB, StdCtrls, FileCtrl,

  {$IFDEF WIN32}
    BDE;
  {$ELSE}
    DbiProcs, DbiTypes, DbiErrs;
  {$ENDIF }


type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Database1: TDatabase;
```

```
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    function ChkPath : Boolean;
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

function TForm1.ChkPath : Boolean;
var
  s : array[0..100] of char;
begin
  If DirectoryExists(Edit1.Text) then begin
    DataBase1.DatabaseName:= 'TempDB';
    DataBase1.DriverName:= 'Standard';
    DataBase1.LoginPrompt:= false;
    DataBase1.Connected := False;
    DataBase1.Params.Add('Path=' + Edit1.Text);
    DataBase1.Connected := TRUE;
    Result := TRUE;
  end
  else begin
    StrPCopy(s,'Directory : ' + Edit1.text + ' Does Not Exist');
    Application.MessageBox(s, 'Error!', MB_ICONSTOP);
    Result := FALSE;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if ChkPath then
    Check(DbiAcqPersistTableLock(Database1.Handle,
                'PARADOX.DRO','PARADOX'));
end;

end.
```

Synchronize a DLL to an Open Dataset

NUMBER : 3105
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 21, 1996

TITLE : Synchronize a DLL to an Open Dataset

Synchronize a DLL to an Open Dataset

This document demonstrates how to use Object Pascal to link a DLL dynamically, on-the-fly, into an active Database, thus, giving the Developer the ability to 'Modularize' features. (Whether at Run-Time or Design-Time)

The technique of linking a DLL Dynamically into an EXE is useful in many cases. Examples include, an Accounting package which offers several 'plug-in' modules (A/R, A/P, General Ledger, etc.) or a Point of Sale package with Current Stock, FIFO/LIFO Ordering, Vendor Tracking, etc., modules.

This TI will provide a solid example of how to do this with one dll, 'Editdll.dll', but will give the Developer the essentials leading to extensive modularation in projects.

Prerequisites:

Familiarity with the TTable component, DLL usage, BDE API, and BDE hCursor knowledge. *WIN API for Dynamical loading of any DLL's.

SAMPLE APPLICATION

The following form, EditForm, is based on the COUNTRY table in the DBDEMO's directory. When the user presses the 'Edit' button or double-clicks a record (row), a dialog box appears, from 'EditDll.dll', displaying record specific information. At this point, the DLL has synchronized itself not only with the dataset (& session) but also with the current record. This mean's, the user can now modify the same data EditForm is viewing! So, with no ado, let's delve into the sample code. (For best results, simply cut and paste into appropriate files)

>Main Form Project<

```
{ MAINDB.DPR }
program maindb;

uses
  Forms,
  mainform in 'mainform.pas' {DBMainForm};

{$R *.RES}
```

```
begin
  Application.Initialize;
  Application.CreateForm(TDBMainForm, DBMainForm);
  Application.Run;
end.

>>

{ MAINFORM.PAS }
unit mainform;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  StdCtrls, Forms, DBCtrls, DB, DBGrids, DBTables, Grids, ExtCtrls,
  BDE;

type
  TDBMainForm = class(TForm)
    Table1Name: TStringField;
    Table1Capital: TStringField;
    Table1Continent: TStringField;
    Table1Area: TFloatField;
    Table1Population: TFloatField;
    DBGrid1: TDBGrid;
    DBNavigator: TDBNavigator;
    Panel1: TPanel;
    DataSource1: TDataSource;
    Panel2: TPanel;
    Table1: TTable;
    EditButton: TButton;
    procedure FormCreate(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DBGrid1DblClick(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  DBMainForm: TDBMainForm;

implementation

{$R *.DFM}

procedure TDBMainForm.FormCreate(Sender: TObject);
begin
  Table1.Open;
end;

// {NOTES: DBHandle is a Handle to the Database & DSHandle is a cursor
//   to the record being viewed. Also, if the purpose is to dynamically
```

```
//   load a DLL at run-time, use the LoadLibrary, GetProcAddress, and
//   FreeLibrary API calls in place of the implicit load calls at startup.
//   An Example of the APIcalling convention is: }
// Type
//   {For GetProcAddress}
//   BDEDataSync =
//      function(const DBHandle: HDBIDB; const DSHandle: HDBICur): Boolean;
//                  stdcall;
//   {For the trapping of DLL load error's}
//   EDLLLoadError = class(Exception);
// var h: hwnd;
//       p: BDEDataSync;
//       LastError: DWord;
// begin
// UpdateCursorPos;
// Try
//    h := loadLibrary('EDITDLL.DLL');
//    {NOTE to Delphi 1.0 users: Whereas Win32 LoadLibrary returns a
//     NULL if the DLL load was unsuccessful, thus requiring a call to
//     GetLastError to find the error, Win16 LoadLibrary returns
//     an error value (less than HINSTANCE_ERROR) which can be checked
//     in the Win16API SDK as to the reason for the failure.}
//    if h = 0 then begin
//         LastError := GetLastError;
//         Raise EDLLLoadError.create(IntToStr(LastError) +
//                                    ': Unable to load DLL');
//         end;
//    try
//        p := getProcAddress(h, 'EditData');
//        if p(DBHandle, Handle) then Resync([]);
//    finally
//        freeLibrary(h);
//    end;
// Except
//    On E: EDLLLoadError do
//       MessageDLG(E.Message, mtInformation, [mbOk],0);
// end;
// end;
// {or}
function EditData(const DBHandle: HDBIDB; const DSHandle: HDBICur):
    Boolean;   stdcall external 'EDITDLL.DLL' name 'EditData';

procedure TDBMainForm.EditButtonClick(Sender: TObject);
begin
  with Table1 do
  begin
    UpdateCursorPos;
// Call the EditData Procedure from EditDll.dll.
    if EditData(DBHandle, Handle) then Resync([]);
  end;
end;

procedure TDBMainForm.DBGrid1DblClick(Sender: TObject);
begin
  EditButton.Click;
end;
```

```
end.

>>

>EDIT DLL PROJECT<
------------------

{ EDITDLL.DPR }
library editdll;

uses
  SysUtils,
  Classes,
  editform in 'editform.pas' {DBEditForm};

exports
  EditData;

begin
end.

>>

{ EDITFORM.PAS }
unit editform;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  StdCtrls, Forms, DBCtrls, DB, DBTables, Mask, ExtCtrls, BDE;

type
  TTableClone = class;

  TDBEditForm = class(TForm)
    ScrollBox: TScrollBox;
    Label1: TLabel;
    EditName: TDBEdit;
    Label2: TLabel;
    EditCapital: TDBEdit;
    Label3: TLabel;
    EditContinent: TDBEdit;
    Label4: TLabel;
    EditArea: TDBEdit;
    Label5: TLabel;
    EditPopulation: TDBEdit;
    DBNavigator: TDBNavigator;
    Panel1: TPanel;
    DataSource1: TDataSource;
    Panel2: TPanel;
    Database1: TDatabase;
    OKButton: TButton;
  private
    TableClone: TTableClone;
```

```
  end;

{ TTableClone }

  TTableClone = class(TTable)
  private
    SrcHandle: HDBICur;
  protected
    function CreateHandle: HDBICur; override;
  public
    procedure OpenClone(ASrcHandle: HDBICur);
  end;

function EditData(const DBHandle: HDBIDB; const DSHandle: HDBICur):
                          Boolean; stdcall;

var
  DBEditForm: TDBEditForm;

implementation

{$R *.DFM}

{ Exports }

function EditData(const DBHandle: HDBIDB; const DSHandle: HDBICur):
                        Boolean; stdcall;
var
  DBEditForm: TDBEditForm;
begin
  DBEditForm := TDBEditForm.Create(Application);
  with DBEditForm do
  try
// Set the handle of the Database1 to that of the currently opened database
    Database1.Handle := DBHandle;
    TableClone := TTableClone.Create(DBEditForm);
    try
      TableClone.DatabaseName := 'DB1';
      DataSource1.DataSet := TableClone;
      TableClone.OpenClone(DSHandle);
      Result := (ShowModal = mrOK);
      if Result then
      begin
        TableClone.UpdateCursorPos;
        DbiSetToCursor(DSHandle, TableClone.Handle);
      end;
    finally
      TableClone.Free;
    end;
  finally
    Free;
  end;
end;

{ TTableClone }
```

```
procedure TTableClone.OpenClone(ASrcHandle: HDBICur);
begin
  SrcHandle := ASrcHandle;
  Open;
  DbiSetToCursor(Handle, SrcHandle);
  Resync([]);
end;

function TTableClone.CreateHandle: HDBICur;
begin
    Check(DbiCloneCursor(SrcHandle, False, False, Result));
end;

end.

>>

{ EDITFORM.DFM }
object DBEditForm: TDBEditForm
  Left = 201
  Top = 118
  Width = 354
  Height = 289
  ActiveControl = Panel1
  Caption = 'DBEditForm'
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 13
  object Panel1: TPanel
    Left = 0
    Top = 0
    Width = 346
    Height = 41
    Align = alTop
    TabOrder = 0
    object DBNavigator: TDBNavigator
      Left = 8
      Top = 8
      Width = 240
      Height = 25
      DataSource = DataSource1
      Ctl3D = False
      ParentCtl3D = False
      TabOrder = 0
    end
    object OKButton: TButton
      Left = 260
      Top = 8
      Width = 75
      Height = 25
      Caption = 'OK'
      Default = True
```

```
      ModalResult = 1
      TabOrder = 1
    end
  end
  object Panel2: TPanel
    Left = 0
    Top = 41
    Width = 346
    Height = 221
    Align = alClient
    BevelInner = bvLowered
    BorderWidth = 4
    Caption = 'Panel2'
    TabOrder = 1
    object ScrollBox: TScrollBox
      Left = 6
      Top = 6
      Width = 334
      Height = 209
      HorzScrollBar.Margin = 6
      HorzScrollBar.Range = 147
      VertScrollBar.Margin = 6
      VertScrollBar.Range = 198
      Align = alClient
      AutoScroll = False
      BorderStyle = bsNone
      TabOrder = 0
      object Label1: TLabel
        Left = 6
        Top = 6
        Width = 28
        Height = 13
        Caption = 'Name'
        FocusControl = EditName
      end
      object Label2: TLabel
        Left = 6
        Top = 44
        Width = 32
        Height = 13
        Caption = 'Capital'
        FocusControl = EditCapital
      end
      object Label3: TLabel
        Left = 6
        Top = 82
        Width = 45
        Height = 13
        Caption = 'Continent'
        FocusControl = EditContinent
      end
      object Label4: TLabel
        Left = 6
        Top = 120
        Width = 22
        Height = 13
```

```
    Caption = 'Area'
    FocusControl = EditArea
  end
  object Label5: TLabel
    Left = 6
    Top = 158
    Width = 50
    Height = 13
    Caption = 'Population'
    FocusControl = EditPopulation
  end
  object EditName: TDBEdit
    Left = 6
    Top = 21
    Width = 135
    Height = 21
    DataField = 'Name'
    DataSource = DataSource1
    MaxLength = 0
    TabOrder = 0
  end
  object EditCapital: TDBEdit
    Left = 6
    Top = 59
    Width = 135
    Height = 21
    DataField = 'Capital'
    DataSource = DataSource1
    MaxLength = 0
    TabOrder = 1
  end
  object EditContinent: TDBEdit
    Left = 6
    Top = 97
    Width = 135
    Height = 21
    DataField = 'Continent'
    DataSource = DataSource1
    MaxLength = 0
    TabOrder = 2
  end
  object EditArea: TDBEdit
    Left = 6
    Top = 135
    Width = 65
    Height = 21
    DataField = 'Area'
    DataSource = DataSource1
    MaxLength = 0
    TabOrder = 3
  end
  object EditPopulation: TDBEdit
    Left = 6
    Top = 173
    Width = 65
    Height = 21
```

```
          DataField = 'Population'
          DataSource = DataSource1
          MaxLength = 0
          TabOrder = 4
        end
      end
    end
    object DataSource1: TDataSource
      Left = 95
      Top = 177
    end
    object Database1: TDatabase
      DatabaseName = 'DB1'
      LoginPrompt = False
      SessionName = 'Default'
      Left = 128
      Top = 176
    end
end

>>
```

Clean-Boot Delphi 2.0 Installation

NUMBER : 3106
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 20, 1996

TITLE : Clean-Boot Delphi 2.0 Installation

HOW TO INSTALL DELPHI IN A CLEAN-BOOT ENVIRONMENT IN WINDOWS 95

CONTENTS

Section 1: INTRODUCTION

This document demonstrates how to do a clean-boot installation for both desktop and Client/Server versions of Delphi 1.0, and the Desktop, Developer, and Client/Server versions of Delphi 2.0 in Windows 95.

Section 2: WHAT IS SAFE MODE?

An important feature that comes with Windows 95 is the "Safe Mode" option. "Safe Mode" is one of the several different ways of loading Windows 95. In "Safe Mode", Windows 95 does not load and use any of your device drivers installed in your system, including your video card, sound card, modem, monitor, mouse, joysticks, etc. Instead, Windows 95 loads up and use the default standard VGA mode in 16 colors and a standard mouse and keyboard driver. One of the advantage of using "Safe Mode" is to free up more conventional memorys and remove any device driver that can cause conflicts during the installation process.

SECTION 3: HOW TO BOOT-UP WINDOWS 95 IN SAFE MODE

---

1. When you boot-up your system the first time, press F8 when you see STARTING WINDOWS 95 on the screen. Usually the message display on the top-left hand corner.

2. If you press F8 before Windows 95 starts loading, you will see the Boot-Up Main Option as follow:

1. Normal
2. Loggged
3. Safe Mode
4. Safe Mode With Network Support
5. Step-by-step Confirmation
6. Command Prompt Only
7. Safe Mode Command Prompt Only
8. Previous Version of MS-DOS

NOTE: Some of the options listed above might not be avalibale on your system, for example, "Safe Mode With Network Support" if your system don't have network feature installed. It depends on your system configuration.

3. Select "Safe Mode" from the Main Opinion and Windows 95 will load the operating system.

4. To install Delphi, please refer to Delphi install.txt and readme.txt.

Creating a Wallpaper Using Delphi

NUMBER : 3128
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Creating a Wallpaper Using Delphi

To the wallpaper in Windows 95 you must use the Win32 API function SystemParametersInfo. SystemParametersInfo retrieves and sets system wide parameters including the wallpaper. The code below illustrates setting the wallpaper to the Athena bitmap.

---- Code Follows ------

```
procedure TForm1.Button1Click(Sender: TObject);
var
  s: string;
begin
    s := 'c:\windows\athena.bmp';
    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0, PChar(s), 0)
end;
```

Detecting Windows Shutdown

NUMBER : 3133
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : October 4, 1996

TITLE : Detecting Windows Shutdown

It is possible for you to detect when windows is being shutdown and to prevent this shutdown from within a running Delphi application.

The simple solution is to add an event handler to the main form's OnCloseQuery event. This event handler occurs as a result of a WM_QUERYENDSESSION message which is sent to all running application in Windows when Windows is about to be shut down. The boolean CanClose parameter to this event can be set to True to allow Windows to shut down or, CanClose can be set to False to prevent Windows from shutting down.

The code below illustrates using this event.

----- Code Follows -------

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  // Ask user if shutdown should occur.
  if MessageDlg('Are you sure?', mtConfirmation, mbYesNoCancel, 0) = mrYes
  then CanClose := true       // Allow Windows to shut down.
  else CanClose := false;    // Prevent Windows from shutting down.
end;
```

Returning Default Cursor after Running Queries

NUMBER : 3136
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Returning Default Cursor after Running Queries

Q: Why does the Mouse cursor not change back to an arrow after I do a Query?

A: When performing an open on a query, Delphi will change the cursor for you, even in the middle of an event, such as a button click.The example below will cause the cursor to show as a SQL Hourglass Icon, after you close the showmessage dialog box. The Mouse will behave as if its in the arrow state.

```
// Add to a button click event, the Query used does not
// matter I used
// Select * from Customer (on IBLocal)

with query1 do begin
  close;
  open;
  showmessage(IntToStr(RecordCount));
  end; // with
```

What appears to be happening is that when Delphi tries to change the Cursor back to the Arrow, a new form has already been shown (the showmessage dialog) and so its job is done as the cursor will automaticly be set to the arrow on the showmessage dialog form.

To solve this problem, add a Application.ProcessMessages before showing anew form, this will clear all pending message commands from the message queue, and the Mouse Cursor will appear normal again.

```
// Add to a button click event, the Query used does
// not matter I used
// Select * from Customer (on IBLocal)

with query1 do begin
  close;
  open;
  application.ProcessMessages;          // Add This Line.
  showmessage(IntToStr(RecordCount));
  end; // with
```

you received with the Borland product to which this information pertains.

Dynamic creation and circularly linking forms

NUMBER : 3137
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Dynamic creation and circularly linking forms

Question: How do I make a simple method to rotate between forms?
How do I add my own return results to a ShowModal form?
How do I instantiate forms at runtime?

Answer: The method required is quite simple to implement. For my example I used 3 forms, the Mainform, Form1, and Form2. I placed a button on the Mainform that will bring up Form1, then from that form you could rotate through any number of forms via buttons placed on those forms. For my example, only Form1 and Form2 can be flipped between.

step 1. Places these two lines in the interface section of this Form, which will be refered to as the main form

```
const
  mrNext = 100;
  mrPrevious = 101;
```

step 2. On the main form add a button and add the following block of code into it.

```
var
  MyForm: TForm;
  R, CurForm: Integer;
begin
   R := 0;
   CurForm := 1;
   while R <> mrCancel do begin
     Case CurForm of
       1: MyForm := TForm1.Create(Application);
       2: MyForm := TForm2.Create(Application);
     end;
     try
       R := MyForm.ShowModal;
     finally
       MyForm.Free;
     end;
     case R of
       MrNext : Inc(CurForm);
       MrPrevious : Dec(CurForm);
     end;
         // these 2 lines will make sure we don't go out of bounds
     if CurForm < 1 then CurForm := 2
     else if CurForm > 2 then CurForm := 1;
   end; // while
```

```
end;
```

step 3. Add forms 1 and 2 (and any others you are going to have) to the uses statement for the MainForm.

step 4. On Form1 and Form2 add the MainForm to the uses (so they can see the constants.

step 5. On Form1, Form2 and all subsequent forms add 2 TBitBtn's, labeled Next and Previous. In the OnClick Events for these buttons add the following line of code.

If it's a Next Button add : ModalResult := mrNext;

If it's a Previous Button add : ModalResult := mrPrevious;

Avoid using Resource Heap with Tabbed Notebooks

NUMBER : 3138
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Avoid using Resource Heap with Tabbed Notebooks

Intended Audience:

Anyone who wants to necessarily avoid using User Resource heap in conjunction with TTabbedNotebooks.

Prerequisites:

- Familiarity with the TTabbedNotebook and TTimer components.
- Understanding of Microsoft Windows Architecture; specifically User resources.

Purpose of the TI:

This document demonstrates how to use Object Pascal to help control the number of active handlesWindows, the User Resource heap specifically, keeps track of. Why does this matter? Simplifying things, Windows keeps track of every Focusable Control via a Handle. The issue is,Windows cannot simply maintain an inexhaustible amount of Windows Handles (4 byte pointers) and this is where this TI's sample code will help "ease" the resource load which might restrict the Delphi Developer.

The User DLL actually is the Library which allocates and maintains resources for all Windows and related data structures, including focusable object's and other unmentioned, but nonetheless important USER functions, under Windows. It is this USER DLL resource limitation* which we are temporarily working around. An example of an added resource load is, every control we add to a form, takes up 4 bytes of the USER 64k.

With this in mind, what exactly are we doing? We will be destroying** the Handles which Windows is architecturely designed to remember. By destroying these Handles, thus freeing USER resource drain, it does not mean we need to recreate said objects again. On the contrary, currently built into the VCL is the ability to keep track of said objects, which are in fact pointers to structures. So, knowing the VCL will maintain a Handle and windows will recreate a new Handle as NEEDED, instead of maintaining one permanently, as designed, we can take control of the USER resources manually and "ease" the overall USER resource load.

This TI will demonstrate the freeing of USER Handle resources via Delphi's TTabbedNoteBook (specifically destroying Page Handles),

Delphi's DestroyHandle (TWinControl procedure for removing USER handles), and the Windows API call LockWindowUpdate (Locking unwanted repaints).

The technique of freeing a TTabbedNoteBook Page Handle can be extended to anyTWinControl descendant. TWinControl is the ancestor class which introduces the creating/destroying of Windows Handles; CreateHandle & DestroyHandle.

* 64K for Win3.1 & 64K for Win95 16-bit subsytem alone. For further information, contact Microsoft or look in the MSDN.
** As a side effect of destroying said Handles, the TTabbedNotebook used in this TI will experience faster page movement.

SAMPLE CODE

The following attached events are direct excerpts from a Project with a TTimer,TTabbedNotebook (with multiple pages) and an assorted plethora of controls on each notebook page. (The later is to emphasize the benefits of adding the below code) The attached Event snippit's should reside in the OnTimer event of the TTimer control and the OnChange event of the TTabbedNotebook, respectively. With no further adou, let's try our new code:

<Unit with the TTabbedNotebook and TTimer declared in it>

...
Implementation

```
Type TSurfaceWin = class(TWinControl);

procedure TForm1.Timer1Timer(Sender: TObject);
begin
{This code will update the Form caption with the percentages of
 free SYSTEM, GDI, &USER for Windows.}
caption := 'SYSTEM: ' +
                inttostr(getfreesystemresources(GFSR_SYSTEMRESOURCES)) +
    ' GDI: ' + inttostr(getfreesystemresources(GFSR_GDIRESOURCES)) +
    ' USER: ' + inttostr(getfreesystemresources(GFSR_USERRESOURCES));
end;

procedure TForm1.TabbedNotebook1Change(Sender: TObject; NewTab:
              Integer;   var AllowChange: Boolean);
begin
{LockWindowUpdate prevents any drawing in a given window}
LockWindowUpdate(handle);

{The reason for TSurfaceWin is because the DestroyHandle call is
declared abstract in TWinControl, which means, only descendant
classes can surface this Procedure. The rest of the line is meant
to flag the current page of the TabbedNotebook and destroy its
handle as we move to another page. NOTE: Even though we destroy
the handle, Windows itself remembers the page object and will
reassign/create a new one when the tab is once more clicked to. }
TSurfaceWin(TabbedNotebook1.pages.objects[tabbedNotebook1.
            pageindex]).DestroyHandle;
```

```
{Release the Lock on the Form so any Form drawing can work}
LockWindowUpdate(0);
end;
```

The DocInput Object: Properties and Methods

NUMBER : 3144
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : The DocInput Object: Properties and Methods

The DocInput Object is an object from the NetManage Internet Solutions Pack deployed with Delphi 2.01. It descibes the input information for a document being transferred to the control. All of the internet controls in the suite that can receive a doc use this object as a property. The DocInput Object has the following properties:
BytesTotal, BytesTransferred, DocLink, FileName, Headers, PushStreamMode, State and Suspended.

BytesTotal is the total byte count of the item to be transmitted. The default or initial value is zero. The data type is a Long. It is a runtime and read only property. This value is obtained from the header property content-length. This value is used by the control to determine the amount of information to be transferred. It is also accessible to you to manage the buffer that you will use to re-assemble the data after transfer.

The BytesTransferred property is a property given to you inside the OnDocInput event. It is a runtime read only property that is of type long. It is set to zero when a new transfer begins. It is updated at the beginning of the OnDocInput Event. This value will reflect the value of the last transfer when no other transfer is in progress. The BytesTransferred property can be used to show progress on a progress bar or to confirm that the actual amount transferred corresponds to that which is expected.

The DocLink property tells the receiving control that the source of the document will not be a sent via data streaming or via an input file. It references a DocOutput.DocLink property which becomes the source of data in the transfer. This property is a read/write property that is runtime only. The property is of type DocLink. It is a string type and the defaultvalue is ''. When the DocLink property is set to a value other than '', the FileName property is automatically set to ''. This property is used to specify a source that is an internet control that has a DocOutput.DocLink property set to correspond with it (i.e. they are used in pairs).

The FileName property is a read/write runtime only property that is of type string. It' default value is''. It must be a valid filename. This property can be set by passing it as an argument

to a DocInput object. If this property is set to a value other than '', then the DocLink property is set to ''.

The Headers property is a runtime readonly property. The "headers"is a collection of DocHeader items that define the doc being transferred. The contents of the headers property should be modified before calling the GetDoc method. Each DocHeader represents a MultiPurpose Internet Mail Extension(MIME). Mime is the mechanism for specifying and describing the format of Internet Message Bodies. (See rfc1341 for details). The headers used depend on the protocol used but two are common to all protocols:

1. content-type
   content type indicates the MIME specification of the ensuing document. "text/plain" is an example of this.
2. content-length
   content length indicates the size of the documents in bytes.

The state property is a runtime read only property of the enumerated type DocStateConstants. The default value is icDocNone. The state property is updated by the control itself each time the DocInput event is activated.

The suspended property is a runtime read only property that is of type boolean. It is set by calling the suspend method. If it is set to true transfer is suspended.

The PushStream property is a read/write, runtime only property that is of the type boolean. The default value is false. If the FileName or DocLink properties are set to values other than '' then the PushStream property is not accessible.

The DocInput object has 4 methods:
GetData, PushStream, SetData and Suspend.

The GetData method tells the DocInput object to retrieve the current block of data when the DocOutput event is activated. This method can only be called during the OnDocInput event and only when the State property is set to icDocData(3). When using the FileName or DocLink properties this method may be used to examine data during transfer.

The PushStream method should only be called when the PushStreamMode has been set to true and when data is available. PushStream sets the State property based on the next step of the document transfer and activates the DocInput event when appropriate. It then returns to wait for the next call to PushStream. SetData chould be called before calling PushStream.

The SetData method specifies the next data buffer to be transferred when the DocInput event is activated. SetData is called during a DocInput event or before calling SendDoc. If it is used before calling SendDoc then it is an alternative to sending the InputData parameters to InputData. The type should be specified as a variant.

The Suspend method takes the form suspend(true) or suspend(false). If the method has been called true twice then it must be called false

twice to resume transfer.

This is some example code on how to use the DocInput Object.
The full program that this code is from can be seen in the demos
sub directory of the Delphi 2.01 CD-Rom. The project name is
SimpMail.dpr. This example is a great exapmle of using the headers
property of the object. The DocInput event also shows proper use
and testing of the State property.

```
{Clear and repopulate MIME headers, using the component's DocInput
property. A separate DocInput OLE object could also be used.
See RFC1521/1522 for completeinformation on MIME types.}
procedure TMainForm.CreateHeaders;
begin
  with SMTP1 do
  begin
    DocInput.Headers.Clear;
    DocInput.Headers.Add('To', eTo.Text);
    DocInput.Headers.Add('From', eHomeAddr.Text);
    DocInput.Headers.Add('CC', eCC.Text);
    DocInput.Headers.Add('Subject', eSubject.Text);
    DocInput.Headers.Add('Message-Id', Format('%s_%s_%s',
      [Application.Title,DateTimeToStr(Now), eHomeAddr.Text]));
    DocInput.Headers.Add('Content-Type',
      'TEXT/PLAIN charset=US-ASCII');
  end;
end;

{Send a simple mail message}
procedure TMainForm.SendMessage;
begin
  CreateHeaders;
  with SMTP1 do
    SendDoc(NoParam, DocInput.Headers, reMessageText.Text, '', '');
end;

{Send a disk file. Leave SendDoc's InputData parameter blank and
specify a filename for InputFile to send the contents of a disk file.
You can use the DocInput event and GetData methods to do custom encoding
(Base64, UUEncode, etc.) }
procedure TMainForm.SendFile(Filename: string);
begin
  CreateHeaders;
  with SMTP1 do
  begin
    DocInput.Filename := FileName;
    SendDoc(NoParam, DocInput.Headers, NoParam, DocInput.FileName, '');
  end;
end;

{The DocInput event is called each time the DocInput state changes
during a mail transfer. DocInput holds all the information
about the current transfer, including the headers, the
number of bytes transferred, and the message data itself. Although not
shown in this example, you may call DocInput's SetData method if
DocInput.State = icDocData to encode the data before each block is
```

```
        sent.}
procedure TMainForm.SMTP1DocInput(Sender: TObject;
  const DocInput: Variant);
begin
  case DocInput.State of
    icDocBegin:
      SMTPStatus.SimpleText := 'Initiating document transfer';
    icDocHeaders:
      SMTPStatus.SimpleText := 'Sending headers';
    icDocData:
      if DocInput.BytesTotal > 0 then
        SMTPStatus.SimpleText :=
       Format('Sending data: %d of %d bytes (%d%%)',
            [Trunc(DocInput.BytesTransferred), Trunc(DocInput.BytesTotal),
             Trunc(DocInput.BytesTransferred/DocInput.BytesTotal*100)])
      else
        SMTPStatus.SimpleText := 'Sending...';
    icDocEnd:
      if SMTPError then
        SMTPStatus.SimpleText := 'Transfer aborted'
      else
        SMTPStatus.SimpleText := Format('Mail sent to %s
      (%d bytes data)',
      [eTo.Text,Trunc(DocInput.BytesTransferred)]);
  end;
  SMTPStatus.Update;
end;
```

Creating Class Properties

NUMBER : 3150
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : October 21, 1996

TITLE : Creating Class Properties

This TI demonstrates how to add a class property to a new component like the Font property on most components.

The following example declares a class, TMyClassProp which contains several fields including an enumerated type field.

Notes:
1. PropertyObject - must be of type TPersistant or a descendant class of TPersistant.
2. Must create an instance of this TMyClassProp in the Create of the component.

----- Unit Follows ------

```
unit SubClass;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs;

type
    { First Create a enumerated list of elements
      then create a set of these elements }

  TEnumList = (Mom, Dad, Sibling, Sister, Brother);
  TEnum = set of TEnumList;

    { Here is the Class Object we want to use as a property in our
      Component.

      It has 4 properties, a Boolean, Word, String, and the
        enumerated type. }

  TMyClassProp = class(TPersistent)
    FBool: Boolean;
    FWord: Word;
    FString: String;
    FEnum:   TEnum;

    published
      property HaveCar: Boolean read FBool Write FBool;
      property Age: Word read FWord write FWord;
      property Name: String read FString write FString;
```

```
        property Relation: TEnum read FEnum write FEnum;
    end; // TMyClassProp


  { Now create the component which will contain a property of type
        TMyClassProp.}
  TMyComponent = class(TComponent)
    FEnum:   TEnum;              { Enumerated type, just for fun }
    FSubClass: TMyClassProp; { The Class Property we want }
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Relation: TEnum read FEnum write FEnum;
    property Relative: TMyClassProp read FSubClass write FSubClass;
    { Published declarations }
  end;


procedure Register;

implementation

  { Override Create, to create an instance of the Class property.
        This is required. }

constructor TMyComponent.Create(AOwner: TComponent);
begin
  inherited;
  FSubClass := TMyClassProp.Create;
end;

 { Override Destroy to perform house cleaning. }
destructor TMyComponent.Destroy;
begin
  FSubClass.Free;
  inherited;
end;

procedure Register;
begin
  RegisterComponents('Samples', [TMyComponent]);
end;

end.
```

Optimizing Oracle Connections with Windows 95

NUMBER : 3151
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Optimizing Oracle Connections with Windows 95

Intended Audience:

Anyone who wants to Improve Oracle IP Connection time under Windows 95.

Prerequisites:

- Windows 95
- Oracle Client Software Installed & Setup to connect to a Oracle Server via TCP/IP.
- (Optional) Delphi 2.0 C/S Software to test result.

Purpose of the TI:

To help speed up Connection time to Oracle under Windows 95. Under WinNT this is not an issue, therefore, this TI applies only to Windows 95. Performance times, as seen on some computer's, with & without this modification are:

Before : Win95 = 10-15 Seconds.
WinNT = 2-3 Seconds.

After : Win95 = 3-4 Seconds. (Big Improvement)

Problem: Windows 95 inherently searches for IPC addresses on some network nodes PRIOR to making a connection to a Oracle DNS whereas WinNT does not.

Solution: Modify the Oracle SQLNET.ORA file to disable said Windows 95 feature.

Step-by-Step Solution:
---
1) Open SQLNET.ORA via Notepad or Write.
(This file can be found in the <ORA_HOME>\network\admin directory. Disregard any other occurrences of this file)

This files looks like this:

```
#################
# Filename.....: sqlnet.ora
# Node.....: local.world
# Date.....: 24-MAY-94 13:23:20
#################
TRACE_LEVEL_CLIENT = OFF
sqlnet.expire_time = 15
names.default_domain = borland.world
```

name.default_zone = borland.world

2) Add the following Parameter to the SQLNET.ORA file:

AUTOMATIC_IPC = OFF

Modified, the SQLNET.ORA file looks like this:

```
#################
# Filename......: sqlnet.ora
# Node.........: local.world
# Date.........: 24-MAY-94 13:23:20
#################
AUTOMATIC_IPC = OFF
TRACE_LEVEL_CLIENT = OFF
sqlnet.expire_time = 15
names.default_domain = borland.world
name.default_zone = borland.world
```

3) Save the file the new SQLNET.ORA file and voila! Anytime an Oracle Connection is initiated, performance with increase from 15 seconds to 3 seconds required. This improves Delphi's already blazing speed.

BDE Callbacks to Provide Status on Operations

NUMBER : 3103
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : August 21, 1996

TITLE : BDE Callbacks to Provide Status on Operations

This TI demonstrates using a BDE DbiCallBack function to drive a progress bar during a lengthy batch move operation under Delphi 2.01.

Additional documentation on BDE function calls can be found in the file BDE32.HLP (located in the directory where the 32 bit IDAPI is installed).

When you install a callback function to the BDE, the BDE will "call back" to a function in your application, letting your application know when certain events take place, and in some cases, allow your application to return information back to the BDE.

The BDE defines several callback types that may be installed:

during large batch operations.
requests a response from the caller.

This TI details installing a cbGENPROGRESS callback to drive a progress bar in your application.

To do this, first call the DbiGetCallBack() function to retrieve the handle to any callback that might already be installed (and it's parameters), and save this information to a data structure. Then install your callback, replacing any previous callback installed.

You will need to pass a pointer the data structure containing the information to the previously installed callback to the BDE when you install your callback, so when your callback function is executed, you can call the original callback (if one is installed).

The BDE will call back to your application every so often, reporting either a text message containing the number of records it has processed or how far the batch move has progressed, expressed as an integer percentage. Your callback should be able to deal with either of these situations. If the percentage field of the callback structure is greater than -1, then the percentage is correct and you are free to update your progress bar. If the percentage reported is less than zero, the callback has received a text message in the szTMsg field containing a message that

includes the number of records processed. In this case, you will need to parse the text message and convert the remaining string to an integer, then calculate the percentage done and update the progress bar.

Finally, when your batch move is complete, you will need to un-register your callback, and reinstall the previous callback function (if it exists).

The following code example is based on a form containing two tables, a batch move component, a progress bar, and a button.

----- Code Follows ---------

```
unit Testbc1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables, ComCtrls;

type
  TForm1 = class(TForm)
    Table1: TTable;
    BatchMove1: TBatchMove;
    Table2: TTable;
    Button1: TButton;
    ProgressBar1: TProgressBar;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Bde; {Dbi Types and Procs are located here}

{$R *.DFM}

{data structure   type to save previous callback info}
type TDbiCbInfo = record
  ecbType       : CBType;
  iClientData : longint;
  DataBuffLn    : word;
  DataBuff      : pCBPROGRESSDesc;
  DbiCbFn       : pointer;
end;
type PDbiCbInfo = ^TDbiCbInfo;
```

```
{Our callback   function}
function DbiCbFn(ecbType        : CBType;
                 iClientData : Longint;
                 CbInfo         : pointer): CBRType stdcall;
var
   s : string;
begin
 {Check to see if the callback type is what we expect}
   if ecbType = cbGENPROGRESS then begin
    {if iPercentDone is less that zero then extract the number}
    {of records processed from szMsg parameter}
     if pCBPROGRESSDesc(cbInfo).iPercentDone < 0 then begin
        s := pCBPROGRESSDesc(cbInfo).szMsg;
        Delete(s, 1, Pos(': ', s) + 1);
      {Calculate percentage done and set the progress bar}
        Form1.ProgressBar1.Position :=
          Round((StrToInt(s) / Form1.Table1.RecordCount) * 100);
     end else
     begin
      {Set the progress bar}
        Form1.ProgressBar1.Position :=
          pCBPROGRESSDesc(cbInfo).iPercentDone;
     end;
   end;
 {was there a previous callback registered}
 {if so - call it and return}
   if PDbiCbInfo(iClientData)^.DbiCbFn <> nil then
      DbiCbFn :=
        pfDBICallBack(PDbiCbInfo(iClientData)^.DbiCbFn)
          (ecbType,
           PDbiCbInfo(iClientData)^.iClientData,
           cbInfo) else
     DbiCbFn := cbrCONTINUE;
end;


procedure TForm1.Button1Click(Sender: TObject);
var
   CbDataBuff: CBPROGRESSDesc; {DBi Structure}
   OldDbiCbInfo : TDbiCbInfo;   {data structure to save previous
                                 callback info}
begin
 {Make sure the table we are moving from is open}
   Table1.Open;
 {make sure the table we are batch moving to is closed}
   Table2.Close;
 {get info about any installed callback}
   DbiGetCallBack(Table2.Handle,
                  cbGENPROGRESS,
                  @OldDbiCbInfo.iClientData,
                  @OldDbiCbInfo.DataBuffLn,
                  @OldDbiCbInfo.DataBuff,
                  pfDBICallBack(OldDbiCbInfo.DbiCbFn));
 {register our callback}
   DbiRegisterCallBack(Table2.Handle,
```

```
                          cbGENPROGRESS,
                          longint(@OldDbiCbInfo),
                          SizeOf(cbDataBuff),
                          @cbDataBuff,
                          @DbiCbFn);

  Form1.ProgressBar1.Position := 0;
  BatchMove1.Execute;

{if a previous callback exists - reinstall it else}
{unregister our callback}
  if OldDbiCbInfo.DbiCbFn <> nil then
    DbiRegisterCallBack(Table2.Handle,
                          cbGENPROGRESS,
                          OldDbiCbInfo.iClientData,
                          OldDbiCbInfo.DataBuffLn,
                          OldDbiCbInfo.DataBuff,
                          OldDbiCbInfo.DbiCbFn) else
    DbiRegisterCallBack(Table2.Handle,
                          cbGENPROGRESS,
                          longint(@OldDbiCbInfo),
                          SizeOf(cbDataBuff),
                          @cbDataBuff,
                          nil);

{lets show our success!}
  Table2.Open;

end;

end.
```

Hints on Overcoming Installation Problems

NUMBER : 3153
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Hints on Overcoming Installation Problems

Delphi 2.0 Up and Running!

This technical document is designed to assist you in reinstalling Delphi 2.0. If you require additional assistance, please call our "Up And Running" hotline at: (408) 461-9195.

Pre-Installation Recommendations

* Before reinstalling Delphi 2.0, run the uninstall option from the add/remove program icon in the Windows 95 control panel, or select the uninstall icon from the Delphi 2.0 program group under Windows NT.

    * Do not run the uninstall from the Delphi 2.0 CD.

* Before uninstalling and reinstalling Delphi 2.0, shut down the any applications using the Borland Database Engine, and close the Local InterBase Server (if running) by selecting the Local InterBase icon (right click from the system tray in Windows 95) and choose "shutdown".

* Installing Delphi 2.0 on a Windows NT system requires Windows NT version 3.51 or later.

* Before installing Delphi 2.0, it is suggested that you install the latest service pack available for your Windows system. Service packs are available from the Microsoft Corporation. One good source is Microsoft's World Wide Web site on the Internet located at www.microsoft.com.

* If you already have a Delphi 1.0 installation on your system, you should install Delphi 2.0 to a different directory structure. The only directory that is recommended to be common to both versions of Delphi is the IDAPI directory.

* Be sure you have proper administration rights on the machine you are installing to.

* Installing Delphi 2.0 to a network server is not supported.

* To install Delphi 2.0 to a computer that does not have a CD-ROM drive, use a network connection or set up a direct cable connection using a parallel or serial cable, and copy the files in the install directory on the Delphi 2.0 CD to a temporary directory on the computer you wish to install to, then run the setup program from the temporary directory created on the computer without the CD-Rom. After successful installation, you may delete the files you copied to the temporary directory.

* If you are running stacker, rename VSTACKER.386 (located in your windows\system directory) to VSTACKER.$$$. Restart Windows, and run the install program again.

* If you have disabled virtual memory, you need to re-enable it for the installation program allowing for at least 64MB of virtual memory.

* Long files names should be enabled on your operating system.

* If you have installed Paradox, delete all trailing lock files before installing.

Installation Errors

---

* If the online registration wizard does not complete your online registration, run the install program again, selecting cancel when you are presented with the choice of registering online. Please complete and mail the registration card included with your Delphi 2.0 product.

* Should you receive a blank dialog box during install, select cancel and the install should continue successfully.

* During the installation, should you receive one of the following error messages:

"out of disk space",
"no temp var",
"error 101"
"error 102"

Add more disk space and/or make sure you have a "temp" environment variable defined and a temp directory created. There must be ample free space on the drive the temp directory is located on.

* During the installation, should you receive the error:

  "Install Shield error filename -51"

  Try the following:

  1) Copy all the files (EXCEPT CTL3D32.DLL) from the runimage\delphi20\windows\system32 directory on the Delphi 2.0 CD to a temporary directory on your hard disk.

  2) Clear the read only attribute on all the files you copied to the temporary directory.

  3) Copy the files to your windows\system directory, or windows\system32 on Windows NT systems.

  4) Run setup again.

Other tips and techniques for a successful installation
---

* Try running the install from your hard disk. To do this, uninstall Delphi 2.0, then simply copy the files from the install directory on the Delphi 2.0 CD to a temporary directory on any hard drive, then run the install program from that temporary directory. After successful installation, you may delete the files you copied to the temporary directory.

* Temporarily rename the win.ini file (located in your Windows directory) to win.in$, reboot, and reinstall or rerun Delphi 2.0. If this helps, suspect any programs listed in the run, or load sections of win.ini, or any non standard print drivers.

* Boot with the standard video driver shipped with your Windows system.

* Check for read-only attributes on files in the Windows and windows\system directories.

Post Installation Issues
---

* Should you receive the error message: "odbc is corrupt or not installed correctly" or "BDECFG32.EXE Error" when you try to install a 32 bit ODBC driver to the BDE, you first need to install a 32 bit ODBC manager, available from InterSolv and Microsoft. One good source is Microsoft's World Wide Web site on the Internet located at www.microsoft.com.

* If you install Delphi 1.0 after installing Delphi 2.0, and Delphi 2.0 loads Delphi 1.0's help files, delete any references to the Delphi 1.0 help file in the WINHELP.INI file located in your Windows directory.

<END>

A Better Way To Print a Form

NUMBER : 3155
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : A Better Way To Print a Form

The following TI details a better way to print the contents of a form, by getting the device independent bits in 256 colors from the form, and using those bits to print the form to the printer.

In addition, a check is made to see if the screen or printer is a palette device, and if so, palette handling for the device is enabled. If the screen device is a palette device, an additional step is taken to fill the bitmap's palette from the system palette, overcoming some buggy video drivers who don't fill the palette in.

Note: Since this code does a screen shot of the form, the form must be the topmost window and the whole from must be viewable when the form shot is made.

```
unit Prntit;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Image1: TImage;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
```

```
uses Printers;


procedure TForm1.Button1Click(Sender: TObject);
var
   dc: HDC;
   isDcPalDevice : BOOL;
   MemDc :hdc;
   MemBitmap : hBitmap;
   OldMemBitmap : hBitmap;
   hDibHeader : Thandle;
   pDibHeader : pointer;
   hBits : Thandle;
   pBits : pointer;
   ScaleX : Double;
   ScaleY : Double;
   ppal : PLOGPALETTE;
   pal : hPalette;
   Oldpal : hPalette;
   i : integer;
begin
 {Get the screen dc}
   dc := GetDc(0);
 {Create a compatible dc}
   MemDc := CreateCompatibleDc(dc);
 {create a bitmap}
   MemBitmap := CreateCompatibleBitmap(Dc,
                                       form1.width,
                                       form1.height);
 {select the bitmap into the dc}
   OldMemBitmap := SelectObject(MemDc, MemBitmap);

 {Lets prepare to try a fixup for broken video drivers}
   isDcPalDevice := false;
   if GetDeviceCaps(dc, RASTERCAPS) and
       RC_PALETTE = RC_PALETTE then begin
      GetMem(pPal, sizeof(TLOGPALETTE) +
        (255 * sizeof(TPALETTEENTRY)));
      FillChar(pPal^, sizeof(TLOGPALETTE) +
        (255 * sizeof(TPALETTEENTRY)), #0);
      pPal^.palVersion := $300;
      pPal^.palNumEntries :=
        GetSystemPaletteEntries(dc,
                                0,
                                256,
                                pPal^.palPalEntry);
      if pPal^.PalNumEntries <> 0 then begin
        pal := CreatePalette(pPal^);
        oldPal := SelectPalette(MemDc, Pal, false);
        isDcPalDevice := true
      end else
      FreeMem(pPal, sizeof(TLOGPALETTE) +
              (255 * sizeof(TPALETTEENTRY)));
   end;
```

```
{copy from the screen to the memdc/bitmap}
  BitBlt(MemDc,
         0, 0,
         form1.width, form1.height,
         Dc,
         form1.left, form1.top,
         SrcCopy);

  if isDcPalDevice = true then begin
     SelectPalette(MemDc, OldPal, false);
     DeleteObject(Pal);
  end;

{unselect the bitmap}
  SelectObject(MemDc, OldMemBitmap);
{delete the memory dc}
  DeleteDc(MemDc);
{Allocate memory for a DIB structure}
  hDibHeader := GlobalAlloc(GHND,
                            sizeof(TBITMAPINFO) +
                            (sizeof(TRGBQUAD) * 256));
{get a pointer to the alloced memory}
  pDibHeader := GlobalLock(hDibHeader);

{fill in the dib structure with info on the way we want the DIB}
  FillChar(pDibHeader^,
           sizeof(TBITMAPINFO) + (sizeof(TRGBQUAD) * 256),
           #0);
  PBITMAPINFOHEADER(pDibHeader)^.biSize :=
     sizeof(TBITMAPINFOHEADER);
  PBITMAPINFOHEADER(pDibHeader)^.biPlanes := 1;
  PBITMAPINFOHEADER(pDibHeader)^.biBitCount := 8;
  PBITMAPINFOHEADER(pDibHeader)^.biWidth := form1.width;
  PBITMAPINFOHEADER(pDibHeader)^.biHeight := form1.height;
  PBITMAPINFOHEADER(pDibHeader)^.biCompression := BI_RGB;

{find out how much memory for the bits}
  GetDIBits(dc,
            MemBitmap,
            0,
            form1.height,
            nil,
            TBitmapInfo(pDibHeader^),
            DIB_RGB_COLORS);

{Alloc memory for the bits}
  hBits := GlobalAlloc(GHND,
                       PBitmapInfoHeader(pDibHeader)^.BiSizeImage);
{Get a pointer to the bits}
  pBits := GlobalLock(hBits);

{Call fn again, but this time give us the bits!}
  GetDIBits(dc,
            MemBitmap,
            0,
            form1.height,
```

```
              pBits,
              PBitmapInfo(pDibHeader)^,
              DIB_RGB_COLORS);

{Lets try a fixup for broken video drivers}
  if isDcPalDevice = true then begin
    for i := 0 to (pPal^.PalNumEntries - 1) do begin
      PBitmapInfo(pDibHeader)^.bmiColors[i].rgbRed :=
        pPal^.palPalEntry[i].peRed;
      PBitmapInfo(pDibHeader)^.bmiColors[i].rgbGreen :=
        pPal^.palPalEntry[i].peGreen;
      PBitmapInfo(pDibHeader)^.bmiColors[i].rgbBlue :=
        pPal^.palPalEntry[i].peBlue;
    end;
    FreeMem(pPal, sizeof(TLOGPALETTE) +
            (255 * sizeof(TPALETTEENTRY)));
  end;

{Release the screen dc}
  ReleaseDc(0, dc);
{Delete the bitmap}
  DeleteObject(MemBitmap);

{Start print job}
  Printer.BeginDoc;

{Scale print size}
  if Printer.PageWidth < Printer.PageHeight then begin
    ScaleX := Printer.PageWidth;
    ScaleY := Form1.Height * (Printer.PageWidth / Form1.Width);
  end else begin
    ScaleX := Form1.Width * (Printer.PageHeight / Form1.Height);
    ScaleY := Printer.PageHeight;
  end;


{Just incase the printer drver is a palette device}
  isDcPalDevice := false;
  if GetDeviceCaps(Printer.Canvas.Handle, RASTERCAPS) and
       RC_PALETTE = RC_PALETTE then begin
   {Create palette from dib}
    GetMem(pPal, sizeof(TLOGPALETTE) +
           (255 * sizeof(TPALETTEENTRY)));
    FillChar(pPal^, sizeof(TLOGPALETTE) +
           (255 * sizeof(TPALETTEENTRY)), #0);
    pPal^.palVersion := $300;
    pPal^.palNumEntries := 256;
    for i := 0 to (pPal^.PalNumEntries - 1) do begin
      pPal^.palPalEntry[i].peRed :=
        PBitmapInfo(pDibHeader)^.bmiColors[i].rgbRed;
      pPal^.palPalEntry[i].peGreen :=
        PBitmapInfo(pDibHeader)^.bmiColors[i].rgbGreen;
      pPal^.palPalEntry[i].peBlue :=
        PBitmapInfo(pDibHeader)^.bmiColors[i].rgbBlue;
    end;
    pal := CreatePalette(pPal^);
```

```
    FreeMem(pPal, sizeof(TLOGPALETTE) +
              (255 * sizeof(TPALETTEENTRY)));
    oldPal := SelectPalette(Printer.Canvas.Handle, Pal, false);
    isDcPalDevice := true
  end;

{send the bits to the printer}
  StretchDiBits(Printer.Canvas.Handle,
                0, 0,
                Round(scaleX), Round(scaleY),
                0, 0,
                Form1.Width, Form1.Height,
                pBits,
                PBitmapInfo(pDibHeader)^,
                DIB_RGB_COLORS,
                SRCCOPY);

{Just incase you printer drver is a palette device}
  if isDcPalDevice = true then begin
    SelectPalette(Printer.Canvas.Handle, oldPal, false);
    DeleteObject(Pal);
  end;


{Clean up allocated memory}
  GlobalUnlock(hBits);
  GlobalFree(hBits);
  GlobalUnlock(hDibHeader);
  GlobalFree(hDibHeader);


{End the print job}
  Printer.EndDoc;


end;
```

Passing Multidimensional Arrays as Parameters

NUMBER : 3187
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : January 8, 1997

TITLE : Passing Multidimensional Arrays as Parameters

Passing an array to a procedure or function is straight forward and behaves as expected. However, passing a multi-dimensional array to a function or procedure is not handled in the same way. Consider MyArray to be defined as:

```
var
MyArray: array[1..3, 1..5] of double;
```

And you want to pass it to a procedure called DoSomeThing() defined as:

```
procedure DoSomeThing(MyArray: array of double);
begin
    showmessage( floattostr( MyArray[1 , 1]));
end;
```

One might think a simple statement like DoSomeThing(MyArray); would do the trick. Unfortunately, this is not the case. The statement DoSomeThing(MyArray); will not compile. The compiler sees the two data structures involved as different types - so it will not allow the statement. The DoSomeThing() procedure is expecting an array of doubles, but the example is passing a multi-dimensional array.

Delphi handles multi-dimensional arrays as user defined type, so there is no syntax to tell a procedure that its parameter(s) are multi-dimensional arrays - without declaring a type. Creating a type, and using this type as the parameter is the correct method to pass a multi-dimensional array. We could just pass a pointer to the array, but inside the function, we need to typecast that pointer. What type to cast it as is the next issue. You would have to have the type defined, or declared identically in 2 places. This method just doesn't make sense.

By defining a type, we can change the process to this:

```
type
  TMyArray = array[1..3, 1..5] of double;

var
    MyArray : TMyArray;
```

```
procedure DoSomeThing(MyArray: TMyArray);
begin
    showmessage( floattostr( MyArray[1 , 1]));
end;
```

Now the actual call looks as we expected:

```
DoSomeThing(MyArray);
```

If you want to use the method of passing a pointer, your function might look like this:

```
type
  PMyArray = ^TMyArray;
  TMyArray = array[1..3, 1..5] of double;

var
    MyArray : TMyArray;

procedure DoSomeThing(MyArray: PMyArray);
begin
 showmessage ( floattostr( (MyArray[2,3]) ));
end;
```

Note, under 32 bit version Delphi, you do not need to dereference the MyArray variable inside DoSomeThing(). Under older versions you might have to refer to MyArray as MyArray^.

If you want to pass just a generic pointer, you may not be able to use it directly. You can declare a local variable, and use it. Again, you may need to cast the local variable for older versions of PASCAL. Of course this method does offer more flexibility in data usage.

```
procedure DoSomeThing(MyArray: pointer);
var
  t : ^TMyArray;
begin
  t := MyArray;
  ShowMessage ( FloatToStr( t[2,3]) );
end;
```

Regardless, both calls that use a pointer method, will look something like:

```
MyArray[2, 3] := 5.6;
DoSomeThing(@MyArray);
```

Using InputBox, InputQuery, and ShowMessage

NUMBER : 3157
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : November 13, 1996

TITLE : Using InputBox, InputQuery, and ShowMessage

This function will demonstrate 3 very powerful and useful procedures built into Delphi.

The InputBox and InputQuery both allow user input.

Use the InputBox function when it doesn't matter if the user chooses either the OK button or the Cancel button (or presses Esc) to exit the dialog box. When your application needs to know if the user chooses OK or Cancel (or presses Esc), use the InputQuery function.

The ShowMessage is another simple way of displaying a message to the user.

```
procedure TForm1.Button1Click(Sender: TObject);
var s, s1: string;
    b: boolean;
begin
  s := Trim(InputBox('New Password', 'Password', 'masterkey'));
  b := s <> '';
  s1 := s;

  if b then
    b := InputQuery('Confirm Password', 'Password', s1);

  if not b or (s1 <> s) then
    ShowMessage('Password Failed');
end;
```

Create a new file with the .wav extension.

NUMBER : 3158
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Create a new file with the .wav extension.

This document describes the process for creating added functionality ,that many Delphi users have requested, to the TMediaPlayer. The new functionality is the ability to create a new file with the .wav format when recording. The procedure "SaveMedia" creates a record type that is passed to the MCISend command. There is an appexception that calls close media if any error occurs while attempting to open the specified file. The application consists two buttons. Button1 calls the OpenMedia and RecordMedia procedures in that order.The CloseMedia procedure is called whenever an exception is generated in this application. Button2 calls the StopMedia,SaveMedia, and CloseMedia procedures.

```
unit utestrec;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
 Forms, Dialogs,MPlayer,MMSystem,StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure AppException(Sender: TObject; E: Exception);
  private
    FDeviceID: Word;
    { Private declarations }
  public
    procedure OpenMedia;
    procedure RecordMedia;
    procedure StopMedia;
    procedure SaveMedia;
    procedure CloseMedia;
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}

var
  MyError,Flags: Longint;

  procedure TForm1.OpenMedia;
  var
    MyOpenParms: TMCI_Open_Parms;
    MyPChar: PChar;
    TextLen: Longint;
  begin
    Flags:=mci_Wait or mci_Open_Element or mci_Open_Type;
  with MyOpenParms do
    begin
      dwCallback:=Handle; // TForm1.Handle
      lpstrDeviceType:=PChar('WaveAudio');
      lpstrElementName:=PChar('');
    end;
  MyError:=mciSendCommand(0, mci_Open, Flags,
          Longint(@MyOpenParms));
  if MyError = 0 then
    FDeviceID:=MyOpenParms.wDeviceID;
  end;

  procedure TForm1.RecordMedia;
  var
    MyRecordParms: TMCI_Record_Parms;
    TextLen: Longint;
  begin
    Flags:=mci_Notify;
    with MyRecordParms do
    begin
      dwCallback:=Handle;   // TForm1.Handle
      dwFrom:=0;
      dwTo:=10000;
    end;
    MyError:=mciSendCommand(FDeviceID, mci_Record, Flags,
    Longint(@MyRecordParms));
  end;

  procedure TForm1.StopMedia;
  var
    MyGenParms: TMCI_Generic_Parms;
  begin
  if FDeviceID <> 0 then
    begin
      Flags:=mci_Wait;
      MyGenParms.dwCallback:=Handle;   // TForm1.Handle
      MyError:=mciSendCommand(FDeviceID, mci_Stop, Flags,
      Longint(@MyGenParms));
    end;
  end;

  procedure TForm1.SaveMedia;
```

```
  type      // not implemented by Delphi
    PMCI_Save_Parms = ^TMCI_Save_Parms;
    TMCI_Save_Parms = record
    dwCallback: DWord;
    lpstrFileName: PAnsiChar;   // name of file to save
  end;
var
  MySaveParms: TMCI_Save_Parms;
begin
  if FDeviceID <> 0 then
  begin
      // save the file...
    Flags:=mci_Save_File or mci_Wait;
    with MySaveParms do
      begin
        dwCallback:=Handle;
        lpstrFileName:=PChar('c:\message.wav');
      end;
    MyError:=mciSendCommand(FDeviceID, mci_Save, Flags,
    Longint(@MySaveParms));
  end;
end;

procedure TForm1.CloseMedia;
var
  MyGenParms: TMCI_Generic_Parms;
begin
  if FDeviceID <> 0 then
  begin
    Flags:=0;
    MyGenParms.dwCallback:=Handle; // TForm1.Handle
    MyError:=mciSendCommand(FDeviceID, mci_Close, Flags,
    Longint(@MyGenParms));
    if MyError = 0 then
      FDeviceID:=0;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenMedia;
  RecordMedia;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  StopMedia;
  SaveMedia;
  CloseMedia;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;
```

```
  procedure TForm1.AppException(Sender: TObject; E: Exception);
  begin
    CloseMedia;
  end;

end.
```

Using MS Internet Explorer 3.0 in Delphi 2

NUMBER : 3159
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Using MS Internet Explorer 3.0 in Delphi 2

Topic: Access Violation when using MS Internet Explorer 3.0 WebBrowser as an OCX in Delphi.

Problem: When you create an OCX wrapper class in Delphi to host the Internet Explorer 3.0 HTML viewer control (named TExplorer or TWebBrowser depending on the age of your IE installation) and use it in a Delphi app that calls the Navigate method of that OCX control, you'll get an access violation as well as possibly ruin your whole Win95 OLE session.

Reason: IE 3.0 calls the IOleClientSite.GetContainer method of Delphi's OCX wrapper implementation. Delphi returns an error code E_NOTIMPL, but IE 3.0 only looks for error code E_NOINTERFACE. IE 3.0 ignores all other error codes and plows ahead with using the bogus interface pointer, thus the access violation occurs.

Solution: In Delphi 2.0's OleCtrls.pas, modify method TOleClientSite.GetContainer to return E_NOINTERFACE instead of E_NOTIMPL as its function result. Note that this doesn't entirely solve the IE 3.0 error checking problem, but it at least placates it.

Important Note: Delphi Developer and Delphi C/S customers can make the change and recompile without affecting any other units. Delphi Desktop customers don't have the VCL source code, so they will need an updated DCU from Borland in order to fix it.

Special Thanks: Danny Thorpe

BDE and Database Desktop Locking Protocol

NUMBER : 3160
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 20, 1996

TITLE : BDE and Database Desktop Locking Protocol

Intended Audience
This information will be of benefit to anyone considering designing a database application using Delphi and the BDE.

Prerequisites
A basic knowledge or interest in Paradox locking protocols and table formats.

Purpose
To give users a better understanding of the table locking protocol.

Table and field types and features supported
Each major release of Paradox has implemented improvements to table structures since version 2.0. All Paradox table types from Paradox 1.0 through Paradox 3.5 are compatible with each other.

Paradox 4.0 adds a new data type to the table format: Binary Large Objects, commonly known as BLObs and new types of Secondary Indices. Paradox 4.0 supports two types of BLOb fields: Memo, and BLOb. Versions of Paradox prior to 4.0, or the Engine prior to 3.0, cannot read, write, or create this new table format. If you attempt to read or write a Paradox 4.0 table type in an earlier version of Paradox, it will return an error that the table is password protected.

Paradox 5.0 added several new data types to the table format: Long Integer, Time, TimeStamp, Logical, Autoincrement, BCD, Bytes. Paradox 7.0 added a descending secondary index. Any table created or modified to include any of these newer features will default to the respective table level. The default table type created using the Database Desktop and the BDE (Borland Database Engine) is a Paradox 4.0. Although the default can be changed in the BDE configuration utility or the Database desktop configuration utility to default to Paradox 3, 4, 5 and 7 for the BDE.

Paradox 4.0 can read, write, and create Paradox table types that are compatible with Paradox 1.0 through Paradox 4.0. So a table created in Paradox 1.0 is compatible with Paradox 4.0. A table created in Engine 1.0 or 2.0 can be read by or written to by Paradox 4.0.

Paradox and the Engine do not change the table type when reading or writing. The table type is only changed when a Restructure is performed on the table.

Paradox Locking Protocols

There are two different Paradox locking protocols: the protocol introduced with Paradox 2.0 and the protocol introduced with Paradox 4.0. These two protocols are not compatible with each other. The locking protocol has no bearing on which type of table a program can work with. There are a few programs that can support either locking protocol; however, these programs can only support one protocol at a time. We will only focus on the 4.0 locking protocol.

Database Desktop/ Paradox 4.0 Locking Protocol

The Paradox 4.0 locking protocol is the only protocol available for Paradox 4.0 and the IDAPI Engine. The designation "Paradox 4.0 locking protocol" represents this style of locking.

Directory Locks

Paradox 4.0 places a locking file, PDOXUSRS.LCK, in each directory where tables are being accessed. This locking file regulates concurrent access to files in the directory. The lock file references PDOXUSRS.NET, so every user must map to the data directory the same way. It also places an exclusive PARADOX.LCK file in the directory as well. It does this to prevent versions of Paradox or the Engine using the older locking system from inadvertently accessing tables.

Working/Shareable Directories

When Paradox or Database Desktop needs to access tables in a directory, they place a "Shareable" PDOXUSRS.LCK file in that directory and an "Exclusive" PARADOX.LCK file in that directory. This designation means that other Paradox 4.0 users can access tables in that directory. The exclusive PARADOX.LCK file is placed in this directory to keep the older, incompatible locking protocol from putting data at risk. In Paradox, this is known as a "Working" directory.

Private/Exclusive Directories

Paradox and Database Desktop also need a directory to store temporary files, such as the Answer table from a query. When Paradox or Paradox Runtime start, they also place an "Exclusive" PDOXUSRS.LCK file in a directory and an "Exclusive" PARADOX.LCK file in the same directory, designating that directory as the location for temporary files. This designation means that other Paradox users cannot access tables in that directory. In Paradox this is known as a "Private" directory.

Table Locks

Paradox 4.0 places each table lock in the directory locking file: PDOXUSRS.LCK. It no longer uses the separate table lock file of previous versions. For example, if three users are viewing the CUSTOMER.DB table and one user is restructuring the ORDERS.DB table, the PDOXUSRS.LCK file will have a shareable lock listed for each of those three users who are viewing the CUSTOMER.DB

table, and an exclusive lock on ORDERS.DB for the user who is restructuring that table.

Paradox 4.0 Locking Protocol Concurrency

In a multi-user environment, the Paradox 4.0 locking protocol maintains concurrency, the simultaneous use of applications, through the PDOXUSRS.NET file. All users who want to share Paradox tables must map to the same PDOXUSRS.NET file in the same way using the same path, but not necessarily the same drive letter. Paradox places a PDOXUSRS.LCK and an exclusive PARADOX.LCK file in each directory where tables are being accessed preventing previous versions of Paradox from accessing files in the same directory. Each user who wants to share tables in that directory must map that directory the same way using the same drive letter and path. Paradox then places all of the locking information for that table in the PDOXUSRS.LCK file, reducing the number of files needed.

Network Control File

The Paradox network control file, PDOXUSRS.NET, serves as the reference point for all lock files created by Paradox. The net file contains the users currently using the BDE and which table they're accessing. Each lock file references the network control file and contains information regarding the locks on the table and by which user, so each user must map to the same network control file in the same way, but not necessarily with the same drive letter.

For example, if you are using volume DATA on server SERVER_1 and the network control file is in the directory \PDOXDATA each user must map \\SERVER_1\DATA:\PDOXDATA the same way, however, each user should, but is not required to use the same drive letter. If the network you are using does not have volumes, then DATA would be a directory off the root of SERVER_1.

If you are mapping \\SERVER_1\DATA to the root of drive P: then each Paradox system would specify the location of PARADOX.NET as P:\PDOXDATA\. However, other users could map \\SERVER_1\DATA to the root of drive O: and specify O:\PDOXDATA\ as the location of the network control file.

Configuring 16 bit Database Engine / IDAPI.CFG

The Database Engine configuration file holds the network specific information and the list of database aliases, as well as other information. You can configure IDAPI using the Database Engine configuration program, BDECFG.EXE, to set the location of the network control file. Also add, delete, modify database aliases including which driver or type of alias used and whether IDAPI will share local tables with other programs using the Paradox 4.0 locking protocol as well as some other specifics regarding the tables and how data is displayed.

Local Settings 16 bit

The WIN.INI file holds the locations of the IDAPI.CFG file, the Database Desktop "Working" directory, and the Database Desktop "Private" directory. You can use any text editor to change these

designations in the WIN.INI file. The location of the IDAPI.CFG file is CONFIGFILE=<full drive, path, and file name> or CONFIGFILE01=<full drive, path, and file NAME> in the [IDAPI] group.

The locations of the Database Desktop "Working" and "Private" directories are in the [DBD] group as WORKDIR=<full drive and directory>, and PRIVDIR=<full drive and directory>.

Configuring 32 bit Database Engine / IDAPI32.CFG

The BDE configuration file holds the same information as the Database Engine configuration file. Use the BDE Configuration, BDECFG32.EXE, to configure IDAPI32.CFG. Optionally you can store the information in the registry or in both the registry and IDAPI32.CFG.

Local Settings 32 bit

The registry holds the locations of the IDAPI32.CFG, the "Working" directory, and the "Private" directory. The location of the IDAPI32.CFG file is stored in HKEY_LOCAL_MACHINE\Software\Borland\Database Engine. The value CONFIGFILE01 holds the data containing <full drive, path, and file name>.

The location of the BDE "Working" and "Private" directories are stored in HKEY_CURRENT_USER\Software\Borland\DBD\7.0\Configuration\WorkDir and HKEY_CURRENT_USER\Software\Borland\DBD\7.0\Configuration\PrivDir respectively. Each directory default value stores the data containing <Full drive and directory>.

Accessing a Paradox Table

The BDE will first try to access the PDOXUSRS.NET file. If a PDOXUSRS.NET file is not found, Paradox will create a new PDOXUSRS.NET file and continue with the startup procedure. If the PDOXUSRS.NET file is found but the owner of this net file used a different path, i.e. mapped to the server differently, an exception of "Multiple net files in use" will be raised and the BDE will shutdown. After the net is successfully opened an exclusive lock, PARADOX.LCK, is placed in the temporary, private, directory. If it fails to place the lock the BDE will shut down. This can fail if some other user has an exclusive lock in this directory or the lock files are using different net files. After it secures a directory for private use it will place a shareable PARADOX.LCK in the working directory and now Initialization is complete.

Getting a record member char array into a memo.

NUMBER : 3162
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 22, 1996

TITLE : Getting a record member char array into a memo.

Handling large strings with the 16 bit Delphi product can be difficult. Especially when the strings are part of a record structure and you would like to flow them into a TMemo. This document shows how to create a record structure that has a 1000 character member and still write it out from a memo then read it back into a memo. The main thrust of the method is to use the GetTextBuf method of the memo. The record structure used is just a string and the array of 1000 chars, but it could be much more complex.

```
unit URcrdIO;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls,dbtables;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
type
  TMyRec = record
    MyArray : array [1..1000] of char;
    mystr : string;
  end;

var
  Form1: TForm1;
  MyRec : TMyRec;
  mylist : TStringlist;
  PMyChar : PChar;
   myfile : file;
   mb : TStream;
implementation
```

```
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);

begin
   assignfile(myfile, 'c:\testblob.txt');
   rewrite(myfile,1);
   fillchar(MyRec.MyArray,sizeof(MyRec.MyArray),#0);
   pmychar:=@MyRec.MyArray;
   StrPCopy(pmychar,memo1.text);
   Blockwrite(MyFile,MyRec,SizeOf(MyRec));
   closefile(MyFile);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   assignfile(myfile, 'c:\testblob.txt');
   reset(myfile,1);
   fillchar(MyRec.MyArray,sizeof(MyRec.MyArray),#0);

   Blockread(MyFile,MyRec,SizeOf(MyRec));
   pmychar:=@MyRec.MyArray;
   Memo1.SetTextBuf(pmychar);

end;

end.
```

How to Get the Most Out of DBDEMOS

NUMBER : 3164
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : December 9, 1996

TITLE : How to Get the Most Out of DBDEMOS

OVERVIEW:

To most Delphi Developers, DBDEMOS is just an alias, but this TI will show that DBDEMOS is more than just sample tables. Included with Delphi 2.0 is the Delphi Demo Explorer where you can easily choose to view a wide range of sample applications, based on concept or by example. Also included in this TI is a directory listing of where to find various other examples and demonstrations included with the Delphi 2.0 CD.

TARGET AUDIENCE:

This TI is aimed at all developers, from beginning to master programmers. Although beginners will be able to run though these examples in order to grasp fundamental Delphi concepts, the master programmers can use this as a reference sheet in order to gain more experience and to research further in any particular area.

DELPHI 2.0 DEMO EXPLORER

The Demo Explorer allows you to browse through a database of all Delphi demonstration programs. As outlined in \info\Borland\Demoexpl\Readdx.Txt on the Delphi 2.0 CD, follow the directions below to install the Delphi 2.0 Demo Explorer:

1. Close Delphi 2.0 if it is running.
2. Create a \DemoExpl directory off of the directory in which you installed Delphi 2.0.
   (i.e., c:\Program Files\Borland\Delphi 2.0\DemoExpl).
3. Copy all of the files from the \info\borland\demoexpl directory of the Delphi 2.0 CD-ROM to the directory you created in step 1.
4. Use the Run option from the Start menu to run the REGEDIT application. Find:
   \HKEY_CURRENT_USER\Software\Borland\Delphi\2.0\Experts
   key by expanding the tree view in the left pane of the Registry Editor window. If you have no experts key then you must create one.
5. Highlight the Experts key in the left pane, and right click on the right pane of the Registry Editor window. From the local menu, select New|String value, enter "DemoExplorer" as the new

value, and press the Enter key.

6. Right click on DemoExplorer in the right pane, and select Modify from the local menu. This will invoke the Edit String dialog. In this dialog, enter the full path from step 1 followed by "\demoexpl.dll". For example: c:\Program Files\Borland\Delphi 2.0\DemoExpl\demoexpl.dll.
7. Demo Explorer is now installed. Run Delphi 2.0, and you can invoke Demo Explorer by selecting "Explore Sample Applications" from the Help menu.

WHERE TO FIND DB DEMOS, DATA AND RELATED FILES
---

1. \Demos\DB contains 17 directories of demonstration database related projects.
2. \Doc
   a) \Dataedit
   b) \DBCal
3. \Demos\Data contains Interbase, Paradox, dBase tables, indexes, validation, memos, bitmaps.
4. \Iblocal\Examples contains Interbase tables Employee.GDB and INTlemp.GDB for Delphi 1.0.
5. \IntrBase\Examples) contains Interbase tables Employee.GDB and INTlemp.GDB for Delphi 2.0.
6. \Source\Samples
   a) Ibctrls.pas
   b) IBEvents.pas
   c) IBProc32.pas

DELPHI 1.0 EXAMPLES
---

1. Animals:
   Simplest possible example using data aware controls and a dBase table. See also FishFact.
2. BDEDLL:
   Demonstrates use of the BDE within a DLL for Delphi 1.0. If the customer is having trouble with a DLL that uses BDE they may check the comments in these two projects. Note: If "DB" appears in the uses clause of a unit, they are using BDE. This is because DB.PAS contains an initialization section that automatically starts the Borland Database Engine. For example, if a data aware control is dropped on the form, DB will be added to the uses clause, and when the application is run, BDE will start, even if the data aware control is removed at a later time.
3. CalcFlds:
   Demonstrates assigning values to calculated fields where looking into a secondary table is necessary.
4. Datalist:
   Populates list boxes with list of available databases, tables, fields and indexes for a given session.
5. DBAware1:
   Demos the use of TDBEdit, TDBComboBox, and TDBListbox.
6. DBAware2:
   Demos the use of TDBLookupCombo and TDBLookupList. The use of TDBLookupCombo and TDBLookupList are commonly

misunderstood - this project shows these controls being used for their intended use.

7. FishFact:
   Simplest possible example using data aware controls and a Paradox table. See also Animals.
8. Format:
   Though not specifically a database application, this project demonstrates the use of the Format function to manipulate the display of floating point numbers.
9. InsQuery:
   Demonstrates inserting and deleting records from a server table using SQL statements containing parameters.
10. LinkQry:
    Displays two grids of master and detail records where the relationship between the master and child tables is performed through a parameterized query.
11. MastApp:
    A complete order entry system with functionality to add, modify, browse and report on orders, customers, parts, and items. Also contains application help, and comments within the main unit on upsizing from the default Paradox to Interbase.
12. QJoin:
    An example of joining two tables using SQL statements. Displays customer and order records where customer numbers match.
13. Range:
    Demonstrates the use of SetRangeStart, SetRangeEnd, ApplyRange, and CancelRange.
14. Search:
    Demonstrates a single search for customer number executed from a button click.
15. Stocks:
    Stock charting application including Market Browser, Industry Charts, Customer Details, Customer Charts, and reporting. The projects demonstrates dynamically population of combo boxes used in turn for dynamically created SQL statements and Chart FX.
16. Tools:
    Review the Readme.txt file in this directory for more information on setting up these projects. Here's an excerpt from the read me:
    * These demos show examples of:
    * An example database MDI application (DBBROWSE.DPR)
    * An example database popup application (SQLMON.DPR)
    * An example of how to derive new database components for the Borland Database Engine system tables (BDETABLES.PAS)
    * An example of how to use the database error stack available in the VCL exception class EDbEngineError. This exception is raised when a Borland Database Engine error occurs (DBEXCEPT.*)
17. TwoForms:
    Shows data aware controls on two forms kept in sync by accessing a common dataset control (a TTable) located on the first form.

DELPHI 2.0 EXAMPLES
---

1. BdQuery:
   Demonstrates how to execute a query in a background thread.
2. CachedUp:
   This example demonstrates how cached updates can be used with live data and in conjunction with the UpdateSQL component for non-live data. See About.txt in this same directory for more information.
3. CsDemos:
   Demonstrates Client/Server concepts, including triggers, stored procedures executed from both TStoredProc and TQuery, and transactions.
4. CtrlGrid:
   A Stock browser showing detail data in a TDBCtrlGrid.
5. DbErrors:
   This example represents a sampling of the way that you might approach trapping a number of database errors. See comments in the data module of this project.
6. Filter:
   Customer/Orders browser allowing a number of query and table based filters. A second form allows building and executing filters over the customer table at runtime.
7. Find:
   Demonstrates record location and incremental searches on filtered and un filtered tables. Includes examples of FindNearest, Locate, and GoToCurrent usage.
8. GdsDemo:
   Order entry application for "Global Diving Supply" with grid and single record views.. This project allows you to experiment with the effects of form inheritance. The project contains two sets of both grid and single record forms which allows you to change the "GDS standard" form and watch the effects on the inherited form. The project also demonstrates techniques for calculating fields, record filtering, and locating records that match the filter, using FindFirst, FindNext, FindPrior, or FindLast.
9. IBDemo:
   Demonstrates listing, registering, and generating InterBase events. The project also includes the TIbEventAlerter component from the samples component palette page.
10. Lookup:
    Example of "lookup" TFields that allow for drop down combos within a TDBGrid. This feature currently available in Delphi 2.0.
11. NavMDI:
    Master and detail data displayed on separate MDI forms, and navigator displayed in the MDI parent.
12. NavSDI:
    Navigator, master and detail each displayed on separate forms.
13. NdxBuild:
    Utility to rebuild indexes for one or all tables within a given alias. Contains code using the BDE API (the dbiRegenIndexes function), application level exception handler, and listing of alias for a given session.
14. QBFDemo:
    This example shows a way to provide users with the ability to define their own queries. If your query returns

records, then a Results Viewer displays those results; otherwise, you'll receive a message indicating that no records matched the search.

Exposing the OnClick of the DBGrid control

NUMBER : 3165
PRODUCT : Delphi
VERSION : 1.0
OS : Windows
DATE : December 9, 1996

TITLE : Exposing the OnClick of the DBGrid control

Many people want to use the OnClick of the TDBGrid. The TDBGrid has no such event. This document tells how to surface the OnClick event for the TDBGrid. The general technique applied here can be used to surface other properties for other objects. If you know that an ancestor could do it this is how to make the descendant do it. One of the powerful things done here is the addition of the csClickEvents to the ControlStyle set property of the control. This allows the control when typecast as THack to recieve and correctly process the click message from windows. The assignment of the OnClick for some other control to the OnClick of the DBGrid1 gives the ability to access and use the OnClick event of a control that has no such event.

This is a hack. There are reasons that dbgrid does not surface the click event. Use this code at your own risk.

```
unit Udbgclk;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs,
  StdCtrls, Grids, DBGrids, DBTables, DB;

type
  thack = class(tcontrol);

  TForm1 = class(TForm)
    DBGrid1: TDBGrid;
    Button1: TButton;
    DataSource1: TDataSource;
    Table1: TTable;
    procedure Button1Click(Sender: TObject);
    procedure FormClick(Sender: TObject);

  private
    { Private declarations }
  public

    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  THack(dbgrid1).controlstyle :=
       THack(dbgrid1).controlstyle + [csClickEvents];
  THack(dbgrid1).OnClick := Form1.OnClick;
end;

procedure TForm1.FormClick(Sender: TObject);
begin
  messagebeep(0);
  application.processmessages;
end;

end.
```

Getting runtime properties at runtime

NUMBER : 3166
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : December 9, 1996

TITLE : Getting runtime properties at runtime

Object Property Information at Runtime.

You may need to know at runtime what properties are available for a particular component at runtime. The list can be obtained by a call to GetPropList. The types, functions and procedures, including GetPropList, that allow access to this property information reside in the VCL source file TYPINFO.PAS.

GetPropList Parameters

```
function GetPropList(TypeInfo: PTypeInfo; TypeKinds: TTypeKinds;
  PropList: PPropList): Integer;
```

The first parameter for GetPropList is of type PTypeInfo, and is part of the RTTI (Run Time Type Information) available for any object. The record structure defined:

```
  PPTypeInfo = ^PTypeInfo;
  PTypeInfo = ^TTypeInfo;
  TTypeInfo = record
    Kind: TTypeKind;
    Name: ShortString;
   {TypeData: TTypeData}
  end;
```

The TTypeInfo record can be accessed through the objects ClassInfo property. For example, if you were getting the property list of a TButton, the call might look, so far, like this:

```
    GetPropList(Button1.ClassInfo, ....
```

The second parameter, of type TTypeKinds, is a set type that acts as a filter for the kinds of properties to include in the list. There are a number of valid entries that could be included in the set (see TYPEINFO.PAS), but tkProperties covers the majority. Now our call to GetPropList would look like:

```
    GetPropList(Button1.ClassInfo, tkProperties ....
```

The last parameter, PPropList is an array of PPropInfo and is defined in TYPEINFO.PAS:

```
  PPropList = ^TPropList;
  TPropList = array[0..16379] of PPropInfo;
```

Now the call might read:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  PropList: PPropList;
begin
  PropList := AllocMem(SizeOf(PropList^));
  GetPropList(TButton.ClassInfo, tkProperties + [tkMethod],
          PropList);
 .
 .
 .
```

Getting Additional Information from the TTYpeInfo Record

The example at the end of this document lists not just the property name, but it's type. The name of the property type resides in an additional set of structures Let's take a second look at the TPropInfo record. Notice that it contains a PPTypeInfo that points ultimately to a TTypeInfo record. TTypeInfo contains the class name of the property.

```
 PPropInfo = ^TPropInfo;
  TPropInfo = packed record
    PropType: PPTypeInfo;
    GetProc: Pointer;
    SetProc: Pointer;
    StoredProc: Pointer;
    Index: Integer;
    Default: Longint;
    NameIndex: SmallInt;
    Name: ShortString;
  end;

  PPTypeInfo = ^PTypeInfo;
  PTypeInfo = ^TTypeInfo;
  TTypeInfo = record
    Kind: TTypeKind;
    Name: ShortString;
   {TypeData: TTypeData}
  end;
```

Example code

The example below shows how to set up the call to GetPropList, and how to access the array elements. TForm will be referenced in this example instead of TButton, but you can substitute other values in the GetPropList call. The visible result will be to fill the list with the property name and type of the

TForm properties.

This project requires a TListBox. Enter the code below in the forms OnCreate event handler.

```
uses TypInfo;

procedure TForm1.FormCreate(Sender: TObject);
var
  PropList: PPropList;
  i: integer;
begin
  PropList := AllocMem(SizeOf(PropList^));
  i := 0;
  try
    GetPropList(TForm.ClassInfo, tkProperties + [tkMethod], PropList);
    while (PropList^[i] <> Nil) and (i < High(PropList^)) do
    begin
      ListBox1.Items.Add(PropList^[i].Name + ': '   +
        PropList^[i].PropType^.Name);
      Inc(i);
    end;
  finally
    FreeMem(PropList);
  end;
end;
```

Search and replace in strings: a task made easy

NUMBER : 3170
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : December 20, 1996

TITLE : Search and replace in strings: a task made easy

Search and replace in strings: a task made easy.

An often overlooked, yet strong feature of Delphi is its history. This product has it roots, and shares some code with Turbo Pascal version 1 released in 1983. Since then, every version of Turbo Pascal added new features, standard functions and procedures to the language. Under its current incarnation, Delphi continues to add features and syntax.

Doing search and replace on strings has been made trivial because of these 3 functions: Pos(), Delete(), and Insert(). Pos() takes two parameters, a pattern search string, and a string to find the pattern in - it returns the location of the string, or 0 if it does not exist. Delete() takes three parameters, the string to delete from, location of where to start deleting, and how much to delete. Similarly, Insert() takes three parameters too. The string that will be inserted, the string to insert into, the location to insert.

Many class properties use strings to store values, so one can use this method on any of them. For instance, the searching and replacing of an entire TMemo component might look like this:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i : integer;
  s1 : string;
  SearchStr : string;
  NewStr : string;
  place : integer;
begin
  SearchStr := 'line';
  NewStr := 'OneEye';
  for i := 0 to Memo1.Lines.Count -1 do begin
    s1 := Memo1.Lines[i];
    Repeat
      Place := pos(SearchStr, s1);
      if place > 0 then begin
        Delete(s1, Place, Length(SearchStr));
        Insert(NewStr, s1, Place);
        Memo1.Lines[i] := s1;
      end; //if-then
```

```
      until place = 0;
   end; //for-loop
end;
```

Dynamically creating a TTable & fields at runtime

NUMBER : 3171
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : April 15, 1997

TITLE : Dynamically creating a TTable & fields at runtime

Delphi allows rapid addition and configuration of database elements to a Delphi project within the design environment, but there are situations where information needed to create and configure objects is not known at design time. For instance, you may want to add the ability to add columns of calculated values (using formulas of the users own creation) to an application at runtime. So without the benefit of the design environment, Object Inspector, and TFields editor, how do you create and configure TFields and other data related components programmatically?

The following example demonstrates dynamically creating a TTable, a database table based off the TTable, TFieldDefs, TFields, calculated fields, and attaches an event handler to the OnCalc event.

To begin, select New Application from the File menu. The entire project will be built on a blank form, with all other components created on-the-fly.

In the interface section of your forms unit, add an OnCalcFields\ event handler, and a TaxAmount field to the form declaration, as shown below.
Later we will create a TTable and hook this handler to the TTable's OnCalcFields event so that each record read fires the OnCalcFields event and in turn executes our TaxAmountCalc procedure.

```
type
  TForm1 = class(TForm)
    procedure TaxAmountCalc(DataSet: TDataset);
  private
    TaxAmount: TFloatField;
  end;
```

In the implementation section add the OnCalc event handler as shown below.

```
procedure TForm1.TaxAmountCalc(DataSet: TDataset);
begin
  Dataset['TaxAmount'] := Dataset['ItemsTotal'] *
    (Dataset['TaxRate'] / 100);
end;
```

Create a OnCreate event handler for the form as shown below (for more information on working with event handlers see the Delphi Users Guide, Chapter 4 "Working With Code").

```
procedure TForm1.FormCreate(Sender: TObject);
var
  MyTable: TTable;
  MyDataSource: TDataSource;
  MyGrid: TDBGrid;
begin

  { Create the TTable component -- the underlying
    database table is created later. }
  MyTable := TTable.Create(Self);
  with MyTable do
  begin

    { Specify an underlying database and table.
      Note: Test.DB doesn't exist yet. }
    DatabaseName := 'DBDemos';
    TableName := 'Test.DB';

    { Assign TaxAmountCalc as the event handler to
      use when the OnCalcFields event fires for
      MyTable. }
    OnCalcFields := TaxAmountCalc;

    { Create and add field definitions to the TTable's
      FieldDefs array, then create a TField using the
      field definition information. }
    with FieldDefs do
    begin
      Add('ItemsTotal', ftCurrency, 0, false);
      FieldDefs[0].CreateField(MyTable);
      Add('TaxRate', ftFloat, 0, false);
      FieldDefs[1].CreateField(MyTable);
      TFloatField(Fields[1]).DisplayFormat := '##.0%';

    { Create a calculated TField, assign properties,
      and add to MyTable's field definitions array. }
      TaxAmount := TFloatField.Create(MyTable);
      with TaxAmount do
      begin
        FieldName := 'TaxAmount';
        Calculated := True;
        Currency := True;
        DataSet := MyTable;
        Name := MyTable.Name + FieldName;
        MyTable.FieldDefs.Add(Name, ftFloat, 0, false);
      end;
    end;

    { Create the new database table using MyTable as
      a basis. }
```

```
    MyTable.CreateTable;
  end;

  { Create a TDataSource component and assign
    to MyTable. }
  MyDataSource := TDataSource.Create(Self);
  MyDataSource.DataSet := MyTable;

  { Create a data aware grid, display on the
    form, and assign MyDataSource to access
    MyTable's data. }
  MyGrid := TDBGrid.Create(Self);
  with MyGrid do
  begin
    Parent := Self;
    Align := alClient;
    DataSource := MyDataSource;
  end;

  { Start your engines! }
  MyTable.Active := True;
  Caption := 'New table ' + MyTable.TableName;
end;
```

The following is the full source for the project.

```
unit gridcalc;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, Grids, DBGrids, ExtCtrls, DBCtrls, DB,
  DBTables, StdCtrls;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure TaxAmountCalc(DataSet: TDataset);
  private
    TaxAmount: TFloatField;
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.TaxAmountCalc(DataSet: TDataset);
begin
  Dataset['TaxAmount'] := Dataset['ItemsTotal'] *
```

```
      (Dataset['TaxRate'] / 100);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  MyTable: TTable;
  MyDataSource: TDataSource;
  MyGrid: TDBGrid;
begin
  MyTable := TTable.Create(Self);

  with MyTable do
  begin
    DatabaseName := 'DBDemos';
    TableName := 'Test.DB';
    OnCalcFields := TaxAmountCalc;

    with FieldDefs do
    begin
      Add('ItemsTotal', ftCurrency, 0, false);
      FieldDefs[0].CreateField(MyTable);
      Add('TaxRate', ftFloat, 0, false);
      FieldDefs[1].CreateField(MyTable);
      TFloatField(Fields[1]).DisplayFormat := '##.0%';
      TaxAmount := TFloatField.Create(MyTable);

      with TaxAmount do
      begin
        FieldName := 'TaxAmount';
        Calculated := True;
        Currency := True;
        DataSet := MyTable;
        Name := MyTable.Name + FieldName;
        MyTable.FieldDefs.Add(Name, ftFloat, 0, false);
      end;
    end;
    MyTable.CreateTable;
  end;

  MyDataSource := TDataSource.Create(Self);
  MyDataSource.DataSet := MyTable;
  MyGrid := TDBGrid.Create(Self);

  with MyGrid do
  begin
    Parent := Self;
    Align := alClient;
    DataSource := MyDataSource;
  end;

  MyTable.Active := True;
  Caption := 'New table ' + MyTable.TableName;
end;

end.
```

Activation and Use of the CPUWindow in the IDE

NUMBER : 3172
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : June 2, 1997

TITLE : Activation and Use of the CPUWindow in the IDE

Warning: The CPU window function is not fully tested and may fail in some cases. If you are having problems with the debugger or your program while using it you should disable it and this may solve your problem. Generally leave it off unless you have a specific need for it.

Delphi 2 has a built in feature, that is off by default, called the CPU window or DisassemblyView. It is easy to use and can be useful in debugging as well as in comparing code for optimization.

To activate this feature, run REGEDIT and edit the registry in the following way.
Go to HKEY_CURRENT_USER\Software\Borland\Delphi\2.0\Debugging.
Once there add a new string key called "ENABLECPU". The value of the new key will be the string "1". That is all it takes.
Now in thge Delphi IDE Select View|CPUWindow.This should bring up the new window.

Now to use this powerful feature to do comparative analysis on 2 snippets that do the same work with different source code use the following procedure.

Create 2 identical event handlers. Inside of each event handler place one of the snippets to be compared. Place a breakpoint on the first line within each handler. Run the application and activate the events. Compare the assembly code for each method. Is one shorter? If so this will execute faster.

Cases worthy of this analysis are lines of code that will execute repeatedly, code that must be optimized for real time application, or code in applications that have to be as fast as possible for whatever reason.

A great example of code that accomplishes the same thing but produces different performance is the "with object do" construct. Many times the source code will be longer using the "with object do" construct but the unassembled code will be shorter. Many times you set properties in a series for dynamically created objects. The code:

```
with TObject.create do
begin
  property1 := ;
```

```
   property2 := ;
   property3 := ;
end;
```

executes faster than:

```
MyObj := TObject.create;
MyObj.Property1 := ;
MyObj.Property2 := ;
MyObj.Property3 := ;
```

Creating a DataAware Control for Browsing Data

NUMBER : 3156
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : November 13, 1996

TITLE : Creating a DataAware Control for Browsing Data

OVERVIEW

This document describes minimal steps necessary to create a data-aware browsing component that displays data for a single field. The example component is a panel with DataSource and DataField properties similar to the TDBText component. See the Component Writer's Guide "Making a Control Data-Aware" for further examples.

WHO SHOULD USE THIS DOCUMENT

To get the best use of this document, you should be familiar with data aware control functionality and fundamental component creation tasks, such as

-- deriving components from existing components

-- overriding constructor and destructors

-- creating new properties

-- getting and setting property values

-- assigning event handlers

BASIC STEPS TO CREATE A DATA-BROWSING COMPONENT

-- Create or derive a component that allows the display, but not the entry of data. For instance, you could use a TMemo with ReadOnly set to true. In the example outlined in this document, we'll use a TCustomPanel. The TCustomPanel will allow display, but not data entry.

-- Add a data-link object to your component. This object manages communication between the component and the database table.

-- Add DataField and DataSource properties to the component.

-- Add methods to get and set the DataField and DataSource.

-- Add a DataChange method the component to handle the

data-link object's OnDataChange event.

-- Override the component constructor to create the datalink
and hook up the DataChange method.

-- Override the component destructor to cleanup the datalink.

CREATING THE TDBPANEL

-- Create or derive a component that allows the display, but
not the entry of data. We'll be using a TCustomPanel as
a starting point for this example.

Choose the appropriate menu option to create a new component (this
will vary between editions of Delphi), and specify TDBPanel
as the class name, and TCustomPanel as the Ancestor type.
You may specify any palette page.

-- Add DB and DBTables to your Uses clause.

-- Add a data-link object to the components private section.
This example will display data for a single field, so we will
use a TFieldDataLink to provide the connection between our new
component and a DataSource. Name the new data-link object
FDataLink.

```
{ example }
private
  FDataLink: TFieldDataLink;
```

-- Add DataField and DataSource properties to the component. We
will add supporting code for the get and set methods in
following steps.

Note: Our new component will have DataField and DataSource
properties and FDataLink will also have its own DataField and
Datasource properties.

```
{ example }
published
  property DataField: string
    read   GetDataField
    write  SetDataField;
  property DataSource: TDataSource
    read   GetDataSource
    write  SetDataSource;
```

-- Add private methods to get and set the DataField and DataSource
property values to and from the DataField and DataSource for
FDataLink.

```
{ example }
private
  FDataLink: TFieldDataLink;
```

```
    function GetDataField: String;
    function GetDataSource: TDataSource;
    procedure SetDataField(Const Value: string);
    procedure SetDataSource(Value: TDataSource);
    .
    .
  implementation
    .
    .
  function TDBPanel.GetDataField: String;
  begin
    Result := FDataLink.FieldName;
  end;

  function TDBPanel.GetDataSource: TDataSource;
  begin
    Result := FDataLink.DataSource;
  end;

  procedure TDBPanel.SetDataField(Const Value: string);
  begin
    FDataLink.FieldName := Value;
  end;

  procedure TDBPanel.SetDataSource(Value: TDataSource);
  begin
    FDataLink.DataSource := Value;
  end;

-- Add a private DataChange method to be assigned to the
    datalink's OnDataChange event.   In the DataChange method
    add code to display actual database field data provided
    by the data-link object.   In this example, we assign
    FDataLink's field value to the panel's caption.

  { example }
  private
    .
    .
    procedure DataChange(Sender: TObject);

  implementation
    .
    .
  procedure TDBPanel.DataChange(Sender: TObject);
  begin
    if FDataLink.Field = nil then
      Caption := '';
    else
      Caption := FDataLink.Field.AsString;
  end;


-- Override the component constructor Create method.   In the
    implementation of Create, create the FDataLink object, and
    assign the private DataChange method to FDataLink's
```

OnDataChange event.

```
{ example }
public
  constructor Create(AOwner: TComponent); override;
  .
  .
implementation
  .
  .
constructor TMyDBPanel.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FDataLink := TFieldDataLink.Create;
  FDataLink.OnDataChange := DataChange;
end;
```

-- Override the component destructor Destroy method. In the implementation of Destroy, set OnDataChange to nil (avoids a GPF), and free FDatalink.

```
{ example }
public
  .
  .
  destructor Destroy; override;
  .
  .
implementation
  .
  .
destructor TDBPanel.Destroy;
begin
  FDataLink.OnDataChange := nil;
  FDataLink.Free;
  inherited Destroy;
end;
```

-- Save the unit and install the component (see the Users Guide, and the Component Writers Guide for more on saving units and installing components).

-- To test the functionality of the component, add a TTable, TDatasource, TDBNavigator and TDBPanel to a form. Set the TTable DatabaseName and Tablename to 'DBDemos' and 'BioLife', and the Active property to True. Set the TDatasource Dataset property to Table1. Set the TDBNavigator and TDBPanel DataSource property to Datasource1. The TDBpanel DataField name should be set as 'Common_Name'. Run the application and use the navigator to move between records to demonstrate the TDBPanel's ability to detect the change in data and display the appropriate field value.

FULL SOURCE LISTING

```
unit Mydbp;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, DB, DBTables;

type
  TDBPanel = class(TCustomPanel)
  private
    FDataLink: TFieldDataLink;
    function GetDataField: String;
    function GetDataSource: TDataSource;
    procedure SetDataField(Const Value: string);
    procedure SetDataSource(Value: TDataSource);
    procedure DataChange(Sender: TObject);
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property DataField: string
      read    GetDataField
      write   SetDataField;
    property DataSource: TdataSource
      read    GetDataSource
      write   SetDataSource;
  end;

  procedure Register;

implementation

  procedure Register;
  begin
    RegisterComponents('Samples', [TDBPanel]);
  end;

  function TDBPanel.GetDataField: String;
  begin
    Result := FDataLink.FieldName;
  end;

  function TDBPanel.GetDataSource: TDataSource;
  begin
    Result := FDataLink.DataSource;
  end;

  procedure TDBPanel.SetDataField(Const Value: string);
  begin
    FDataLink.FieldName := Value;
  end;

  procedure TDBPanel.SetDataSource(Value: TDataSource);
  begin
    FDataLink.DataSource := Value;
```

```
  end;

  procedure TDBPanel.DataChange(Sender: TObject);
  begin
    if FDataLink.Field = nil then
      Caption := ''
    else
      Caption := FDataLink.Field.AsString;
  end;

  constructor TDBPanel.Create(AOwner: TComponent);
  begin
    inherited Create(AOwner);
    FDataLink := TFieldDataLink.Create;
    FDataLink.OnDataChange := DataChange;
  end;

  destructor TDBPanel.Destroy;
  begin
    FDataLink.Free;
    FDataLink.OnDataChange := nil;
    inherited Destroy;
  end;

end.
```

Steps for FAT32 Support with the BDE

NUMBER : 3188
PRODUCT : BDE
VERSION : 3.0
OS : Windows
DATE : April 3, 1997

TITLE : Steps for FAT32 Support with the BDE

With the current build of Windows 95, 950B, a new file system is now supported, FAT32. The FAT file system had problems with large cluster sizes with large hard disks making storage inefficient. FAT32 keeps the cluster size at 4k enabling efficient data storage.

COMMON BDE ERRORS EXPERIENCED WITH FAT32 SYSTEMS

"Invalid File Name"
"Invalid Table Name"
"File Not Found"
"Table Not Found"

NOTE: All Delphi and CBuilder exceptions are of type EDBEngine error. The errors above are displayed within this exception type.

To determine if you have a FAT32 system:
1) Right click the mouse on the My Computer icon on the desktop. Select the Properties menu item. On the general tab, under System, it will say 4.00.950 B. NOTE: 4.00.950A or 4.00.950 does not support FAT32. At the time this was written 4.00.950B is the latest. Microsoft may decide to create newer builds with FAT32 support.

2) Double click on the My Computer icon on the desktop. Right click on each local drive and select the Properties menu item. If any of the drive Types is Local Disk (FAT32), then you have a FAT32 partition on your system. Note: Local Disk (FAT) is not FAT32.

If in the above steps you have determined that you have a FAT32 file system, there are two ways to get the appropriate patches:
1) Internet (Web). Go to http://www.borland.com/techsupport/bde/utilities.html and get the file named 'BDE v3.5 32-Bit core DLLs including FAT32 enhancement (~ 2.4M)'. Unzip and install the new BDE.

2) CompuServe. Go to the BDEVTOOLS forum, Borland DB Engine section. Get files DEMO35D.ZIP and FAT32.ZIP. Unzip and install the BDE 3.5. After installation, unzip the FAT32.ZIP file and place the IDAPI32.DLL over the existing

IDAPI32.DLL.

OTHER OPTIONS/WORKAROUNDS IF UPGRADE IS NOT AN OPTION

If for any reason, the new DLL or set of DLLs cannot be placed on the system, you must follow the proceeding criteria:

1) All sub-directories must be 8 characters or less.
2) If tables must be in a 9 character or greater sub-directory, you can still open the table by
   a) (Delphi, CBuilder) specifying the full path and file name in TTable.Table name, leaving DatabaseName blank.
   b) (BC++) Passing NULL to the pszAliasName parameter and using the full path and file name in DbiOpenTable. NOTE: DbiSetDirectory will not work.
3) Table names can be greater than 8 characters with spaces.

Implementing Drag and Drop Functionality

NUMBER : 3192
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 4, 1997

TITLE : Implementing Drag and Drop Functionality

This document describes the process of creating a drag and drop functionality that could be applied to text items and graphical objects. Regular text indicates the procedural steps to execute to achieve the functionality. Text in the parentheses is explanation and background information.

1. Place the two edit boxes onto your form. {One will serve as the source for the text that you will drag and drop. The other will serve as the destination you will drop the text onto.}

2. Select the first edit box and name it SourceEdit by entering that name in the name property in the Object Inspector.

3. Select the second edit box and name it SenderEdit. {The names source and sender are not necessary. They are suggested because "source" and "sender" are default variables given to work with in the OnDragOver and OnDragDrop event procedure code blocks. The source variable is where the drag operation began, and the sender variable is the control that was dropped onto. See the help topics related to messages for further information on this and related topics.}

4. Click the SourceEdit component so it is selected and go to the Object Inspector. Set its DragMode property to dmAutomatic.

5. Select the SenderEdit and go to the events page of the Object Inspector and double click on the OnDragOver event.

6. Type exactly what is in the quotes here where the cursor is between the begin and end statements: "Accept := True;". {Notice that the Accept variable is supplied for you by default inside of the OnDragOver event procedure code block.}

7. Now go to the events page of the Object Inspector and double click on the OnDragDrop event.

8. Type exactly what is in the quotes here where the cursor is between the begin and end statements:
" SenderEdit.Text := SourceEdit.Text ".

Now if you run the application you will be able to click down on the SourceEdit component and drag and drop onto the SenderEdit component. When you drop onto the SenderEdit it

changes its text to whatever was in the SourceEdit.

Trapping Windows Messages in Delphi

NUMBER : 3194
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1997

TITLE : Trapping Windows Messages in Delphi

While Delphi provides many ways to trap the incoming messages to VCL controls, you may require a quick and effective method of "stealing" and replacing the windows procedure of a given window. The method presented here will effectively trap all Windows messages for any window or VCL control that has a Window handle property.

Background:
For every window that is created, a structure is created by the system that holds information about the window. The information contained in this structure includes, among other things, the window's parent, instance information, and the address of the window's main window procedure. It is through this procedure that all Windows messages are sent for that window.

The Microsoft Windows API provides functions to both retrieve and set the values contained in this structure, making it possible to retrieve the address of a window's main procedure and reset that address to point to a user installed function.

To replace the windows procedure, we will use the API function SetWindowsLong(), and pass in the handle of the window we wish to work with, the constant GWL_WNDPROC (telling the function we wish to replace the windows procedure), and the address of our function we wish to replace it with.

When we call the SetWindowsLong() function, it will return the previous value we are replacing. We will want to remember that value so we can call the original procedure for all the messages we do not care to trap. We will also want to reinstall the old procedure when we are done, so that message handling returns to normal.

In our example code, we will trap the WM_VSCROLL message of a TDbGrid component, giving us an indication that the user has scrolled the vertical scroll bar. Since all messages for the control are passed first through our procedure, we can effectively trap and do processing for any event before it is ever received by the component.

For a list of other notification messages you may be interesed in trapping, you may search the Delphi's Windows help file for messages starting with WM_.

Also note that the code has been written to compile

under both sixteen and thirty-two bit versions of Delphi.

```
unit WinProc1;

interface

uses
{$IFDEF WIN32}
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, DB, DBTables;
{$ELSE}
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, DB, DBTables, Grids, DBGrids;
{$ENDIF}

type
  TForm1 = class(TForm)
    DBGrid1: TDBGrid;
    Table1: TTable;
    DataSource1: TDataSource;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

type
{$IFDEF WIN32}
  WParameter = LongInt;
{$ELSE}
  WParameter = Word;
{$ENDIF}
  LParameter = LongInt;

{Declare a variable to hold the window procedure we are replacing}
var
  OldWindowProc : Pointer;

function NewWindowProc(WindowHandle : hWnd;
                       TheMessage    : WParameter;
                       ParamW        : WParameter;
                       ParamL        : LParameter) : LongInt
{$IFDEF WIN32} stdcall; {$ELSE} ; export; {$ENDIF}
begin

{ Process the message of your choice here }
```

```
   if TheMessage = WM_VSCROLL then begin
      ShowMessage('The vertical scrollbar is scrolling!');
   end;

{ Exit here and return zero if you want        }
{ to stop further processing of the message }

{ Call the old Window procedure to }
{ allow processing of the message. }
   NewWindowProc := CallWindowProc(OldWindowProc,
                                   WindowHandle,
                                   TheMessage,
                                   ParamW,
                                   ParamL);
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
{ Set the new window procedure for the control }
{ and remember the old window procedure.           }
   OldWindowProc := Pointer(SetWindowLong(DbGrid1.Handle,
                                          GWL_WNDPROC,
                                          LongInt(@NewWindowProc)));
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
{ Set the window procedure back }
{ to the old window procedure.    }
   SetWindowLong(DbGrid1.Handle,
                 GWL_WNDPROC,
                 LongInt(OldWindowProc));

end;

end.


(*
{ The program's main source file }
program WinProc;

uses
   Forms,
   WinProc1 in 'WinProc1.pas' {Form1};

{$R *.RES}

begin
{$IFDEF WIN32}
   Application.Initialize;
{$ENDIF}
   Application.CreateForm(TForm1, Form1);
   Application.Run;
end.
```

*)

{ end of ti }

Moving Projects Between Machines or Directories

NUMBER : 3214
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : July 7, 1997

TITLE : Moving Projects Between Machines or Directories

Moving Projects Between Machines

The main files involved in a project are the DPR, DFM, PAS, and RES. These are the main files in any project. There are other files associated with any project. This document focuses on these because they are the minimum files you need to move a project.

Moving a project between machines is easy if you follow a few simple rules. First keep all the files associated with the project in the same directory. This allows for quick movement all in one operation. To move the files copy the entire directory to a new temporary directory. Now delete all the files except those with these extensions DPR, PAS, DFM, and RES. All the other files will be recreated during the first build. Having a single directory eliminates questions as to which version of a dcu or pas file is going into the build. If third party components are in the project you should install those on to the Delphi Component Palette of the target machine before attempting to open the project on that machine.

If the project files were not all in one directory you may end up editing the DPR file to provide the appropriate new path information on the target machine. The text involved is in the uses clause of the DPR or main project file. You may also choose to preserve the DSK files or the DOF files if you have a specific need to preserve your desktop setttings or your project options settings respectively. The options file can be important in cases where you have specific unit aliases, you have specified directories under project options, use packages, version info, run parameters, and version info. The dsk file is less critical and generally not moved with a project.

The res file should be moved with the project in cases where your project uses custom cursors, icons, string resources, bitmaps, etc. These are resources included by you with the compiler directive to be linked into the project and become part of the executable. The sting resources are used for internationalization or localization. Other custom resource are visual enhancment of the project user interface.

For more detailed information on the specific file types referred to in this document see the TI on file types, TI 3213.

Creating a form based on a string

NUMBER : 3197
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1997

TITLE : Creating a form based on a string

Creating a form based on a string.

OVERVIEW

This document demonstrates how to instantiate a Delphi form based on a string which specifies the name of the type. Sample code is given.

WHO SHOULD USE THIS DOCUMENT

Anyone with a basic familiarity with Delphi programming. Applies to any version of Delphi.

INSTANTIATING A FORM BASED ON A STRING

To be able to instantiate forms based on the string representing the names of their type, you must first register the type with Delphi. This is accomplished with the function "RegisterClass".
RegisterClass is prototyped as follows:

```
procedure RegisterClass(AClass: TPersistentClass);
```

AClass is a class of TPersistent. In other words, the class you are registering must be descended at some point from TPersistent. Since all Delphi controls, including forms, fit this requiremnet, we have will have no problem. This could not be used, for instance, to register classes descended directly from TObject.

Once the class has been registered, you can find a pointer to the type by passing a string to FindClass. This will return a class reference, which you can use to create the form.
An example is in order:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  b : TForm;
  f : TFormClass;
begin
  f := TFormClass(findClass('Tform2'));
  b := f.create(self);
  b.show;
```

```
end;
```

This will create the TForm2 type that we registered
with RegisterClass.

WORKING SAMPLE PROGRAM

Create a new project, and then add 4 more forms, for
a total of 5. You can populate them with controls if
you like, but for this example, it is not important.

On the first form, put down an edit control, and
a pushbutton. Take all forms, but the main form,
out of the AutoCreate List.
Finally, paste the following code over the code
in unit1, this will give you the ability to type
the Class NAME into the Edit, and it will create
that form for you.

```
unit Unit1;

interface

uses
  Unit2, Unit3, Unit4, Unit5, Windows, Messages,
  SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  RegisterClass(Tform2);
  RegisterClass(Tform3);
  RegisterClass(Tform4);
  RegisterClass(Tform5);
```

```
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   f : Tformclass;
begin
   f := tformclass(findClass(edit1.text));
   with f.create(self) do
    show;
end;
```

Finding the color depth of a canvas

NUMBER : 3198
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 3, 1997

TITLE : Finding the color depth of a canvas

When programming graphics in Windows, it it often desirable to know the number of colors available for a given canvas. This information is available through the GetDeviceCaps() functon.

The color depth of a canvas is calculated by multiplying the number of bits per pixel that is required for a given canvas by the number of planes the canvas uses.

You can find the color depth of a canvas in by using the following code:

```
TotalNumBitsPerPixel :=
    GetDeviceCaps(Canvas.Handle, BITSPIXEL) *
    GetDeviceCaps(Canvas.Handle, PLANES)
```

Will give you the total number of bits used to color a pixel.

Return values of:

1 = 2 colors (monochrome)
4 = 16 colors
8 = 256 colors
15 = 32,768 colors
16 = 65,536 colors
24 = 16,777,216 colors

Using FindFirst and the WIN_32_FIND_DATA structure

NUMBER : 3199
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1997

TITLE : Using FindFirst and the WIN_32_FIND_DATA structure

Here is how to access the WIN_32_FIND_DATA structure of a TSearchRec using the FindFirst(), FindNext(), and FindClose() functions.

In this example, we will show how to display both the long and short filename for each file found. Notice in the example that the unit name is fully qualified for the FindFirst(), FindNext(), and FindClose() functions, since there is more than one implementation for these function names. The implementations that we are calling are the Delphi wrapper functions to the WinApi functons.

```
var
  sr : TSearchRec;
  R : integer;
begin
   R := Sysutils.FindFirst('C:\*.*', faAnyFile, sr);
   while R = 0 do
   begin
     Memo1.Lines.Add(sr.FindData.cFileName);
     if sr.FindData.cAlternateFileName <> '' then
       Memo1.Lines.Add(sr.FindData.cAlternateFileName) else
       Memo1.Lines.Add(sr.FindData.cFileName);
     R := Sysutils.FindNext(sr);
   end;
   Sysutils.FindClose(sr);
end;
```

<end of ti>

Setting the pixels per inch property of TPrinter

NUMBER : 3200
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1997

TITLE : Setting the pixels per inch property of TPrinter

When changing printers, be aware that fontsizes may not always scale properly. To ensure proper scaling, set the PixelsPerInch property of the font after changing the printer index property. Be sure not to make the change until you have started the print job.

Here are two examples:

```
*******************************************************
uses Printers;

var
  MyFile: TextFile;
begin
  Printer.PrinterIndex := 2;
  AssignPrn(MyFile);
  Rewrite(MyFile);
  Printer.Canvas.Font.Name := 'Courier New';
  Printer.Canvas.Font.Style := [fsBold];
  Printer.Canvas.Font.PixelsPerInch:=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSY);
  Writeln(MyFile, 'Print this text');
  System.CloseFile(MyFile);
end;

*******************************************************


uses Printers;

begin
  Printer.PrinterIndex := 2;
  Printer.BeginDoc;
  Printer.Canvas.Font.Name := 'Courier New';
  Printer.Canvas.Font.Style := [fsBold];
  Printer.Canvas.Font.PixelsPerInch:=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSY);
  Printer.Canvas.Textout(10, 10, 'Print this text');
  Printer.EndDoc;
end;
```

<end of ti>

How to use a string table resource

NUMBER : 3201
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 3, 1997

TITLE : How to use a string table resource

Stringtable resources are a very useful tool when your application must store a large number of strings for use at runtime. While you may be tempted to directly embed strings into your executable, using a stringtable resource offers two advantages: 1) The strings contained in the stringtable do not consume memory until they are loaded by your application. 2) Stringtables are easily edited, providing an easy path to internationally localized versions of your application.

Stringtables are compiled into a ".res" file that is attached to your application's exe file at build time. Even after you distribute your appliaction, the stringtable contained in your application's exe file can be edited with a resource editor. My favorite resource editor is Borland's Resource Workshop that ships with the RAD pack. It can produce and edit both 16 and 32 bit resources that are self contained, standalone, or embedded in a .exe or .dll in full WYSIWYG fashion.

It's worth noting that all versions of Delphi ship with the Borland Resource Command Line Compiler (BRCC.EXE and BRCC32.EXE), and can be found in Delphi's Bin directory.

For our example, we will build an internationalized application that displays two buttons. The buttons will have captions for "Yes" and "No" presented in English, Spanish, and Swedish.

It's worth noting that if you want to build international applications using Delphi, you should take a look at Borland's Delphi Translation Suite and Language Pack software. These packages can make porting your application to international markets a snap!

Example:

We first must create a text file containing our string resources in the applications build directory. You may name the file anything you wish, so long as it has the file extension ".rc" and the filename without the extension is not the same as any unit or project filename. This is very important, as Delphi also will create a number of resource files for your project automatically.

Here is the contents of the .rc file for our example. It contains the words "Yes" and "No" in English, Spanish, and Swedish:

```
STRINGTABLE
{
 1, "&Yes"
 2, "&No"
 17, "&Si"
 18, "&No"
 33, "&Ja"
 34, "&Nej"
}
```

The file starts with the key word stringtable denoting that a string table resource will follow. Enclosed in the curly braces are the strings. Each string is listed by its index identifier, followed by the actual string data in quotes. Each string may contain up to 255 characters. If you need to use a non-standard character, insert the character as a backslash character followed by the octal number of the character you wish to insert. The only exception is when you want to embed a backslash character, you will need to use two backslashes. Here are two examples:

```
1, "A two\012line string"

2, "c:\\Borland\\Delphi"
```

The Index numbers that you use are not important to the resource compiler. You should keep in mind that string tables are loaded into memory in 16 string segments.

To compile the .rc file to a .res file that can be linked with your application, simply type on the dos command line the full path to the resource compiler, and the full path to the name of the .rc file to compile. Here is an example:

```
c:\Delphi\Bin\brcc32.exe c:\Delphi\strtbl32.rc
```

When the compiler is finished, you should have a new file with the same name as the .rc file you've compiled, only with an extension of ".res".

You can link the resource file with your application simply by adding the following statement to your application's code, substituting the name of your resource file:

```
{$R ResFileName.RES}
```

Once the .res file is linked to your program, you can load the resource from any module, even if you specified the $R directive in the implementation section of a different unit.

Here is an example of using the Windows API function LoadString(), to load the third string contained in a string resource into a character array:

```
  if LoadString(hInstance,
                3,
                @a,
                sizeof(a)) <> 0 then ....
```

In this example, the LoadString() function accepts the hInstance of the module containing the resource, the string index to load, the address of the character array to load the string to, and the size of the character array. The LoadString function returns the number of characters that where actually loaded not including the null terminator. Be aware that this can differ from the number of bytes loaded when using unicode.

Here is a complete example of creating an international application with Borland's Delphi. The application is compatible with both 16 and 32 bit versions of Delphi.

To do this, you will need to create two identical .rc files, one for the 16 bit version, and the other for the 32 bit version, since the resources needed for each platform are different. In this example. we will create one file named STRTBL16.rc and another called STRTBL32.rc. Compile the STRTBL16.rc file using the BRCC.exe compiler found in Delphi 1.0's bin directory, and compile STRTBL32.rc using the BRCC32.exe compiler found in Delphi 2.0's bin directory.

We have taken into account the language that Windows is currently using at runtime. The method for getting this information differs under 16 and 32 bit Windows. To make the code more consistant, we have borrowed the language constants from the Windows.pas file used in 32 bit versions of Delphi.

```
{$IFDEF WIN32}
    {$R STRTBL32.RES}
{$ELSE}
    {$R STRTBL16.RES}
    const LANG_ENGLISH = $09;
    const LANG_SPANISH = $0a;
    const LANG_SWEDISH = $1d;
{$ENDIF}


function GetLanguage : word;
{$IFDEF WIN32}
{$ELSE}
  var
```

```
      s : string;
      i : integer;
{$ENDIF}
begin
{$IFDEF WIN32}
   GetLanguage := GetUserDefaultLangID and $3ff;
{$ELSE}
   s[0] := Char(GetProfileString('intl',
                                 'sLanguage',
                                 'none',
                                 @s[1],
                                 sizeof(s)-2));
   for i := 1 to length(s) do
      s[i] := UpCase(s[i]);
   if s = 'ENU' then GetLanguage := LANG_ENGLISH else
   if s = 'ESN' then GetLanguage := LANG_SPANISH else
   if s = 'SVE' then GetLanguage := LANG_SWEDISH else
      GetLanguage := LANG_ENGLISH;
{$ENDIF}
end;


procedure TForm1.FormCreate(Sender: TObject);
var
   a : array[0..255] of char;
   StrTblOfs : integer;
begin

 {Get the current language and stringtable offset}
   case GetLanguage of
      LANG_ENGLISH : StrTblOfs := 0;
      LANG_SPANISH : StrTblOfs := 16;
      LANG_SWEDISH : StrTblOfs := 32;
    else
      StrTblOfs := 0;
   end;

 {Load language dependent "Yes" and set the button caption}
   if LoadString(hInstance,
                 StrTblOfs + 1,
                 @a,
                 sizeof(a)) <> 0 then
      Button1.Caption := StrPas(a);

 {Load language dependent "No" and set the button caption}
   if LoadString(hInstance,
                 StrTblOfs + 2,
                 @a,
                 sizeof(a)) <> 0 then
      Button2.Caption := StrPas(a);
end;

<end of ti>
```

Borland Assist for Delphi/400

NUMBER : 3202
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : March 10, 1997

TITLE : Borland Assist for Delphi/400

Borland Assist for Delphi/400
Technical Support for U.S. and Canada.

Borland Assist offers online and contract technical support for Delphi/400. You can obtain guidance on product usability, application design, programming, operating systems and hardware compatibility, and issues related to your specific application and data. While Borland does provide guidance in program design, coding, and debugging, Borland engineers will not write or debug user applications specific to your environment. Online resources are intended to provide a forum for sharing technical information with other users and obtaining technical support documentation. Contract support is_for more specific or in-depth user needs.

Choosing the Best Support for Your Needs

Online Support

Borland offers a complete support environment at Borland Online. This service is free to the user and includes Frequently Asked Questions FAQS), discussion forums, bug-submission utilities, a library of product-specific information, links to related resources, and much more.

Borland Online: http://www.borland.com

Telephone Support

Client/Server Developer Assist for Delphi/400:

Client/Server Developer Assist for Delphi/400 is a specialized program for corporate and independent developers, as well as consultants. Under this contract, users receive consultative support including assistance with user-created applications or database design, coding and debugging issues for Client Objects/400 and Screen Designer/400.
Price: $3,500 for one contact; additional contact $3,000

How to Order Borland Assist

When you purchase a Borland Assist contract, you will be issued a Personal Identification Number (PIN) that gives you access to hotline support. You may purchase Borland Assist via check, VISA, MasterCard, American Express, or company purchase order.

To learn more about Borland Assist services or purchase a Borland Assist support contract, please call 1-800-636-7778. Purchase order acceptance is subject to credit approval.

Money-back Guarantee

Getting quality technical support for your Borland products is just a phone call away, and Borland ensures your complete satisfaction via a 30-day, money-back guarantee on any paid subscription you select.

Terms and Conditions

Your use of the Borland Assist Support Program is subject to Borland's terms and conditions, which will be sent to you upon request. Among other things, those terms and conditions provide you with a license to use the work we do for you under the Program (which Borland owns, including copyrights) and include Borland's disclaimer of warranties and limitation of liability. Services subject to change without notice.

TRichEdit Printing in Delphi 2 & Windows NT 4.0

NUMBER : 3204
PRODUCT : Delphi
VERSION : 2.0
OS : Win32
DATE : April 2, 1997

TITLE : TRichEdit Printing in Delphi 2 & Windows NT 4.0

TRichEdit Printing Under Windows NT 4.0

This document provides a Delphi Unit that will solve the "divide by zero" error that occurs when printing from a TRichEdit control under Windows NT 4.0. To use this Unit simply include it in the USES clause of the Unit that you want to print from. Instead of calling the RichEdit.Print() method to print, call the PrintRichEdit() procedure and pass it the TRichEdit control that you want to print as a parameter.

If you own the VCL source code you can make the changes shown below to the TCustomRichEdit.Print method in the COMCTRLS.PAS file.

```
unit PrtRichU;

interface

uses
  ComCtrls;

procedure PrintRichEdit(const Caption: string;
                        const RichEdt: TRichEdit);

implementation
uses
  Windows, RichEdit, Printers;

procedure PrintRichEdit(const Caption: string;
                        const RichEdt: TRichEdit);
var
  Range: TFormatRange;
  LastChar, MaxLen, LogX, LogY, OldMap: Integer;
begin
  FillChar(Range, SizeOf(TFormatRange), 0);
  with Printer, Range do
  begin
    BeginDoc;
    hdc := Handle;
    hdcTarget := hdc;
    LogX := GetDeviceCaps(Handle, LOGPIXELSX);
    LogY := GetDeviceCaps(Handle, LOGPIXELSY);
    if IsRectEmpty(RichEdt.PageRect) then
    begin
      rc.right := PageWidth * 1440 div LogX;
```

```
      rc.bottom := PageHeight * 1440 div LogY;
    end
    else begin
      rc.left := RichEdt.PageRect.Left * 1440 div LogX;
      rc.top := RichEdt.PageRect.Top * 1440 div LogY;
      rc.right := RichEdt.PageRect.Right * 1440 div LogX;
      rc.bottom := RichEdt.PageRect.Bottom * 1440 div LogY;
    end;
    rcPage := rc;
    Title := Caption;
    LastChar := 0;
    MaxLen := RichEdt.GetTextLen;
    chrg.cpMax := -1;
    OldMap := SetMapMode(hdc, MM_TEXT);
    SendMessage(RichEdt.Handle, EM_FORMATRANGE, 0, 0);
    try
      repeat
        chrg.cpMin := LastChar;
        LastChar := SendMessage(RichEdt.Handle, EM_FORMATRANGE, 1,
                                Longint(@Range));
        if (LastChar < MaxLen) and (LastChar <> -1) then NewPage;
      until (LastChar >= MaxLen) or (LastChar = -1);
      EndDoc;
    finally
      SendMessage(RichEdt.Handle, EM_FORMATRANGE, 0, 0);
      SetMapMode(hdc, OldMap);
    end;
  end;
end;

end.
```

How to use a user defined resource.

NUMBER : 3209
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : April 3, 1997

TITLE : How to use a user defined resource.

(*

The following demonstrates linking a user defined resource.

Resources are compiled into a ".res" file that is
attached to your application's .exe file at build time.
Even after you distribute your application, the resources
contained in your application's .exe file can be edited with
a resource editor. The resource editor in Borland's Resource
Workshop, that ships with the RAD pack, can produce and edit
both 16 and 32 bit resources that are self contained,
standalone, or embedded in a .exe or .dll in full WYSIWYG
fashion.

It's worth noting that all versions of Delphi ship with the
Borland Resource Command Line Compiler (BRCC.EXE and BRCC32.EXE),
and can be found in Delphi's Bin directory.

We first must create a text file containing our resource
definitions in the application's build directory. You may
name the file anything you wish, so long as it has the
file extension ".rc" and the filename without the extension
is not the same as any unit or project filename. This is
very important, as Delphi also will create a number of
resource files for your project automatically.

Here is the contents of the .rc file for our example:

MYUSERDATA MYDATATYPE TEST.TXT

MYUSERDATA is the resource name, MYDATATYPE is the resource type
and TEST.TXT is the name of the file continuing the user data we
will link in.

We must next create the TEST.TXT file, using any ASCII editor
(notepad will do fine). For our example, the file needs to
contain one single line:

Hello!

To compile the .rc file to a .res file that can be linked
with your application, simply type on the dos command line
the full path to the resource compiler, and the full path
to the name of the .rc file to compile. Here is an example:

c:\Delphi\Bin\brcc32.exe c:\Delphi\MYRES.RC

When the compiler is finished, you should have a new file
with the same name as the .rc file you've compiled, only
with an extension of ".res".

You can link the resource file with your application simply
by adding the following statement to your application's
code, substituting the name of your resource file:

{$R ResFileName.RES}

where ResFileName.RES is the actual name of the compiled
.res file.

Once the .res file is linked to your program, you can load
the resource from any module, even if you specified the
$R directive in a different unit.

To actually use the resource, you must make a few Windows API
calls. First you will call the FindResource() function, passing
the instance handle of your application, the name of the resource
to load, and the resource type. If FindResource is successful, the
function will return a handle to the unloaded resource.

Next, you can call the SizeOfResource() function to find the aligned
size of the resource, if you want to know the actual size of the
loaded resources memory block. Be forewarned that this not the
actual size of the data, but rather the size of the memory block
that will be allocated when you load the resource. If you want
to know the actual size of the data, you will need to embed that
information in the data itself.

To load the resource, we will call the LoadResource() function,
passing the handle returned to us when we called FindResource().
This function will return a handle to the loaded resource that we
can pass to the LockResource() function to retrieve a pointer to
the resource that we use to actually work with the data.

In our example, we will loop through the data, adding the data
to a string, until we find the embedded '!' character, signaling
that the end of the data has been found.

Finally, we will free the resource by first calling UnLockResource()
and then FreeResource(), passing the handle of the loaded resource.

Example Code:
*)

implementation

{$R *.DFM}

{$R MYRES.RES}

```
procedure TForm1.Button1Click(Sender: TObject);
var
  hRes            : THandle; {handle to the resource}
  pRes            : pointer; {pointer to the resource}
  ResSize         : longint; {aligned size of the resource}
  i               : integer; {counting variable}
{$IFDEF WIN32}
  s               : shortstring; {a string to play with}
{$ELSE}
  s               : string;         {a string to play with}
{$ENDIF}
begin
{find the resource}
  hRes := FindResource(hInstance,
                       'MYUSERDATA',
                       'MYDATATYPE');
  if hRes = 0 then begin
    ShowMessage('Could not find the resource');
    exit;
  end;
{get the aligned size of the resource}
  ResSize := SizeOfResource(hinstance, hRes);
  if ResSize = 0 then begin
    ShowMessage('Nothing to load - size = 0');
    Exit;
  end;
{load the resource}
  hRes := LoadResource(hInstance, hRes);
  if hRes = 0 then begin
    ShowMessage('Resource Load Failure');
    Exit;
  end;
{Get a pointer to the resource}
  pRes := LockResource(hRes);
  if pRes = nil then begin
    ShowMessage('Resource Lock Failure');
    FreeResource(hRes);
    Exit;
  end;

{convert the resource pointer to a string}
  s:='';
  i := 0;
  while pChar(pRes)[i] <> '!' do begin
    s := s + pChar(pRes)[i];
    inc(i);
  end;

{prove it works}
  ShowMessage(s);

{unlock and free the resource}
  UnLockResource(hRes);
  FreeResource(hRes);
end;
```

end.

<end of ti>

A better way to do pointer arithmetic

```
    NUMBER   :   3210
   PRODUCT   :   Delphi
   VERSION   :   All
        OS   :   Windows/Win32
      DATE   :   April 3, 1997

     TITLE   :   A better way to do pointer arithmetic
```

```
(*

This document demonstrates a technique for incrementing pointers under
Borland Pascal 7.0, Turbo Pascal for Windows, and Delphi.

Send it any pointer, and an offset of the pointer you really
want (yes, huge pointers > 64 are supported) and it will return
a pointer to that location. Unlike most other huge pointer code,
this snippet will take into account the fact that you may be
sending a pointer that does not have a zero based offset.

Be aware that you cannot access memory that crosses a 64k
boundary in Win16. If you need to access a record that
spans a 64K boundary, you must first grab the portion of the
record that exists before the boundary, then grab the second
portion that exists above the boundary, and piece them together.

Note the function is designed to be upwardly portable to
Win32 where there is no need to do pointer manipulation
since Win32 uses a flat (non-segmented) memory model where
there are no 64k boundaries. While using the function call
to get a pointer under Win32 is not exactly efficient, it
will make porting legacy 16 bit code to Win32 a breeze!

*)

type
  PtrRec = record
    Lo : Word;
    Hi : Word;
  end;
  PHugeByteArray = ^THugeByteArray;
  THugeByteArray = array[0..0] of Byte;

function GetBigPointer(lp : pointer;
                       Offset : LongInt) : Pointer;

begin
 {$IFDEF WIN32}
  GetBigPointer := @PHugeByteArray(lp)^[Offset];
 {$ELSE}
  Offset := Offset + PTRREC(lp).Lo;
  GetBigPointer := Ptr(PtrRec(lp).Hi + PtrRec(Offset).Hi * SelectorInc,
                       PtrRec(Offset).Lo);
 {$ENDIF}
```

```
end;


{Lets test it!}

procedure TForm1.Button1Click(Sender: TObject);
var
   h   : THandle; {handle to the memory block}
   p   : pointer; {pointer to the memory block}
   p2 : pointer; {pointer for testing}
   p3 : pointer; {pointer for testing}
begin
 {allocate two hundred thousand bytes of memory and zero out}
   h := GlobalAlloc(GHND, 200000);

 {get a pointer to the allocated memory}
   p := GlobalLock(h);

 {get a pointer to the byte at index 75000}
   p2 := GetBigPointer(p, 75000);

 {get a pointer to the byte 80000 bytes from p2}
   p3 := GetBigPointer(p2, 80000);

 {verify the byte p3 points to is zero}
   Memo1.Lines.Add(IntToStr(pByte(p3)^));

 {change the value of the byte p3 points to}
   pByte(p3)^ := 10;

 {verify the change}
   Memo1.Lines.Add(IntToStr(pByte(p3)^));

 {free the memory}
   GlobalUnlock(h);
   GlobalFree(h);
end;

end.
```

<end of ti>

Assuring Proper Font Scaling When Printing

NUMBER : 3211
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : April 9, 1997

TITLE : Assuring Proper Font Scaling When Printing

When changing printers, be aware that fontsizes may not always scale properly. To ensure proper scaling, set the PixelsPerInch property of the font after changing the printer index property. Be sure not to make the change until you have started the print job.

Here are two examples:

```
*******************************************************
uses Printers;

var
  MyFile: TextFile;
begin
  Printer.PrinterIndex := 2;
  AssignPrn(MyFile);
  Rewrite(MyFile);
  Printer.Canvas.Font.Name := 'Courier New';
  Printer.Canvas.Font.Style := [fsBold];
  Printer.Canvas.Font.PixelsPerInch:=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSY);
  Writeln(MyFile, 'Print this text');
  System.CloseFile(MyFile);
end;

*******************************************************


uses Printers;

begin
  Printer.PrinterIndex := 2;
  Printer.BeginDoc;
  Printer.Canvas.Font.Name := 'Courier New';
  Printer.Canvas.Font.Style := [fsBold];
  Printer.Canvas.Font.PixelsPerInch:=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSY);
  Printer.Canvas.Textout(10, 10, 'Print this text');
  Printer.EndDoc;
end;
```

<end of ti>

BDE Error listing

NUMBER : 3212
PRODUCT : BDE
VERSION : 2.x
OS : Windows
DATE : June 19, 1997

TITLE : BDE Error listing

This document lists all the errors that can be returned by the BDE. This information is derived from IDAPI.H (C++) or BDE.INT (C++ Builder and Delphi). You also wish to look at the following document(s) for addition information on specific errors:

TI2733 - Troubleshooting IDAPI Error Messages
TI2814 - Handling EDBEngineError Exceptions
TI3160 - BDE and Database Desktop Locking Protocol

Note: You can use DbiGetErrorString() to retrieve the error string for any error.

System Related (Fatal Error)

8449 : $2101 : Cannot open a system file.
8450 : $2102 : I/O error on a system file.
8451 : $2103 : Data structure corruption.
8452 : $2104 : Cannot find Engine configuration file.
8453 : $2105 : Cannot write to Engine configuration file.
8454 : $2106 : Cannot initialize with different configuration file.
8455 : $2107 : System has been illegally re-entered.
8456 : $2108 : Cannot locate IDAPI32 .DLL.
8457 : $2109 : Cannot load IDAPI32 .DLL.
8458 : $210A : Cannot load an IDAPI service library.
8459 : $210B : Cannot create or open temporary file.

Object of Interest not Found

8705 : $2201 : At beginning of table.
8706 : $2202 : At end of table.
8707 : $2203 : Record moved because key value changed.
8708 : $2204 : Record/Key deleted.
8709 : $2205 : No current record.
8710 : $2206 : Could not find record.
8711 : $2207 : End of BLOB.
8712 : $2208 : Could not find object.
8713 : $2209 : Could not find family member.
8714 : $220A : BLOB file is missing.
8715 : $220B : Could not find language driver.

Physical Data Corruption

8961 : $2301 : Corrupt table/index header.

8962 : $2302 : Corrupt file - other than header.
8963 : $2303 : Corrupt Memo/BLOB file.
8965 : $2305 : Corrupt index.
8966 : $2306 : Corrupt lock file.
8967 : $2307 : Corrupt family file.
8968 : $2308 : Corrupt or missing .VAL file.
8969 : $2309 : Foreign index file format.

I/O related error

9217 : $2401 : Read failure.
9218 : $2402 : Write failure.
9219 : $2403 : Cannot access directory.
9220 : $2404 : File Delete operation failed.
9221 : $2405 : Cannot access file.
9222 : $2406 : Access to table disabled because of previous error.

Resource or Limit error

9473 : $2501 : Insufficient memory for this operation.
9474 : $2502 : Not enough file handles.
9475 : $2503 : Insufficient disk space.
9476 : $2504 : Temporary table resource limit.
9477 : $2505 : Record size is too big for table.
9478 : $2506 : Too many open cursors.
9479 : $2507 : Table is full.
9480 : $2508 : Too many sessions from this workstation.
9481 : $2509 : Serial number limit (Paradox).
9482 : $250A : Some internal limit (see context).
9483 : $250B : Too many open tables.
9484 : $250C : Too many cursors per table.
9485 : $250D : Too many record locks on table.
9486 : $250E : Too many clients.
9487 : $250F : Too many indexes on table.
9488 : $2510 : Too many sessions.
9489 : $2511 : Too many open databases.
9490 : $2512 : Too many passwords.
9491 : $2513 : Too many active drivers.
9492 : $2514 : Too many fields in Table Create.
9493 : $2515 : Too many table locks.
9494 : $2516 : Too many open BLOBs.
9495 : $2517 : Lock file has grown too large.
9496 : $2518 : Too many open queries.
9498 : $251A : Too many BLOBs.
9499 : $251B : File name is too long for a Paradox version 5.0 table.
9500 : $251C : Row fetch limit exceeded.
9501 : $251D : Long name not allowed for this tablelevel.

Integrity Violation

9729 : $2601 : Key violation.
9730 : $2602 : Minimum validity check failed.
9731 : $2603 : Maximum validity check failed.
9732 : $2604 : Field value required.

9733 : $2605 : Master record missing.
9734 : $2606 : Master has detail records. Cannot delete or modify.
9735 : $2607 : Master table level is incorrect.
9736 : $2608 : Field value out of lookup table range.
9737 : $2609 : Lookup Table Open operation failed.
9738 : $260A : Detail Table Open operation failed.
9739 : $260B : Master Table Open operation failed.
9740 : $260C : Field is blank.
9741 : $260D : Link to master table already defined.
9742 : $260E : Master table is open.
9743 : $260F : Detail table(s) exist.
9744 : $2610 : Master has detail records. Cannot empty it.
9745 : $2611 : Self referencing referential integrity must be entered one at a time with no other changes to the table
9746 : $2612 : Detail table is open.
9747 : $2613 : Cannot make this master a detail of another table if its details are not empty.
9748 : $2614 : Referential integrity fields must be indexed.
9749 : $2615 : A table linked by referential integrity requires password to open.
9750 : $2616 : Field(s) linked to more than one master.
9751 : $2617 : Expression validity check failed.

Invalid Request

9985 : $2701 : Number is out of range.
9986 : $2702 : Invalid parameter.
9987 : $2703 : Invalid file name.
9988 : $2704 : File does not exist.
9989 : $2705 : Invalid option.
9990 : $2706 : Invalid handle to the function.
9991 : $2707 : Unknown table type.
9992 : $2708 : Cannot open file.
9993 : $2709 : Cannot redefine primary key.
9994 : $270A : Cannot change this RINTDesc.
9995 : $270B : Foreign and primary key do not match.
9996 : $270C : Invalid modify request.
9997 : $270D : Index does not exist.
9998 : $270E : Invalid offset into the BLOB.
9999 : $270F : Invalid descriptor number.
10000 : $2710 : Invalid field type.
10001 : $2711 : Invalid field descriptor.
10002 : $2712 : Invalid field transformation.
10003 : $2713 : Invalid record structure.
10004 : $2714 : Invalid descriptor.
10005 : $2715 : Invalid array of index descriptors.
10006 : $2716 : Invalid array of validity check descriptors.
10007 : $2717 : Invalid array of referential integrity descriptors.
10008 : $2718 : Invalid ordering of tables during restructure.
10009 : $2719 : Name not unique in this context.
10010 : $271A : Index name required.
10011 : $271B : Invalid session handle.

10012 : $271C : invalid restructure operation.
10013 : $271D : Driver not known to system.
10014 : $271E : Unknown database.
10015 : $271F : Invalid password given.
10016 : $2720 : No callback function.
10017 : $2721 : Invalid callback buffer length.
10018 : $2722 : Invalid directory.
10019 : $2723 : Translate Error. Value out of bounds.
10020 : $2724 : Cannot set cursor of one table to another.
10021 : $2725 : Bookmarks do not match table.
10022 : $2726 : Invalid index/tag name.
10023 : $2727 : Invalid index descriptor.
10024 : $2728 : Table does not exist.
10025 : $2729 : Table has too many users.
10026 : $272A : Cannot evaluate Key or Key does not pass filter condition.
10027 : $272B : Index already exists.
10028 : $272C : Index is open.
10029 : $272D : Invalid BLOB length.
10030 : $272E : Invalid BLOB handle in record buffer.
10031 : $272F : Table is open.
10032 : $2730 : Need to do (hard) restructure.
10033 : $2731 : Invalid mode.
10034 : $2732 : Cannot close index.
10035 : $2733 : Index is being used to order table.
10036 : $2734 : Unknown user name or password.
10037 : $2735 : Multi-level cascade is not supported.
10038 : $2736 : Invalid field name.
10039 : $2737 : Invalid table name.
10040 : $2738 : Invalid linked cursor expression.
10041 : $2739 : Name is reserved.
10042 : $273A : Invalid file extension.
10043 : $273B : Invalid language Driver.
10044 : $273C : Alias is not currently opened.
10045 : $273D : Incompatible record structures.
10046 : $273E : Name is reserved by DOS.
10047 : $273F : Destination must be indexed.
10048 : $2740 : Invalid index type
10049 : $2741 : Language Drivers of Table and Index do not match
10050 : $2742 : Filter handle is invalid
10051 : $2743 : Invalid Filter
10052 : $2744 : Invalid table create request
10053 : $2745 : Invalid table delete request
10054 : $2746 : Invalid index create request
10055 : $2747 : Invalid index delete request
10056 : $2748 : Invalid table specified
10058 : $274A : Invalid Time.
10059 : $274B : Invalid Date.
10060 : $274C : Invalid Datetime
10061 : $274D : Tables in different directories
10062 : $274E : Mismatch in the number of arguments
10063 : $274F : Function not found in service library.
10064 : $2750 : Must use baseorder for this operation.
10065 : $2751 : Invalid procedure name
10066 : $2752 : The field map is invalid.

Locking/Contention related

10241 : $2801 : Record locked by another user.
10242 : $2802 : Unlock failed.
10243 : $2803 : Table is busy.
10244 : $2804 : Directory is busy.
10245 : $2805 : File is locked.
10246 : $2806 : Directory is locked.
10247 : $2807 : Record already locked by this session.
10248 : $2808 : Object not locked.
10249 : $2809 : Lock time out.
10250 : $280A : Key group is locked.
10251 : $280B : Table lock was lost.
10252 : $280C : Exclusive access was lost.
10253 : $280D : Table cannot be opened for exclusive use.
10254 : $280E : Conflicting record lock in this session.
10255 : $280F : A deadlock was detected.
10256 : $2810 : A user transaction is already in progress.
10257 : $2811 : No user transaction is currently in progress.
10258 : $2812 : Record lock failed.
10259 : $2813 : Couldn't perform the edit because another user changed the record.
10260 : $2814 : Couldn't perform the edit because another user deleted or moved the record.

Access Violation - Security related

10497 : $2901 : Insufficient field rights for operation.
10498 : $2902 : Insufficient table rights for operation. Password required.
10499 : $2903 : Insufficient family rights for operation.
10500 : $2904 : This directory is read only.
10501 : $2905 : Database is read only.
10502 : $2906 : Trying to modify read-only field.
10503 : $2907 : Encrypted dBASE tables not supported.
10504 : $2908 : Insufficient SQL rights for operation.

Invalid context

10753 : $2A01 : Field is not a BLOB.
10754 : $2A02 : BLOB already opened.
10755 : $2A03 : BLOB not opened.
10756 : $2A04 : Operation not applicable.
10757 : $2A05 : Table is not indexed.
10758 : $2A06 : Engine not initialized.
10759 : $2A07 : Attempt to re-initialize Engine.
10760 : $2A08 : Attempt to mix objects from different sessions.
10761 : $2A09 : Paradox driver not active.
10762 : $2A0A : Driver not loaded.
10763 : $2A0B : Table is read only.
10764 : $2A0C : No associated index.
10765 : $2A0D : Table(s) open. Cannot perform this operation.
10766 : $2A0E : Table does not support this operation.
10767 : $2A0F : Index is read only.

10768 : $2A10 : Table does not support this operation because it is not uniquely indexed.
10769 : $2A11 : Operation must be performed on the current session.
10770 : $2A12 : Invalid use of keyword.
10771 : $2A13 : Connection is in use by another statement.
10772 : $2A14 : Passthrough SQL connection must be shared

Os Error not handled by Idapi

11009 : $2B01 : Invalid function number.
11010 : $2B02 : File or directory does not exist.
11011 : $2B03 : Path not found.
11012 : $2B04 : Too many open files. You may need to increase MAXFILEHANDLE limit in IDAPI configuration.
11013 : $2B05 : Permission denied.
11014 : $2B06 : Bad file number.
11015 : $2B07 : Memory blocks destroyed.
11016 : $2B08 : Not enough memory.
11017 : $2B09 : Invalid memory block address.
11018 : $2B0A : Invalid environment.
11019 : $2B0B : Invalid format.
11020 : $2B0C : Invalid access code.
11021 : $2B0D : Invalid data.
11023 : $2B0F : Device does not exist.
11024 : $2B10 : Attempt to remove current directory.
11025 : $2B11 : Not same device.
11026 : $2B12 : No more files.
11027 : $2B13 : Invalid argument.
11028 : $2B14 : Argument list is too long.
11029 : $2B15 : Execution format error.
11030 : $2B16 : Cross-device link.
11041 : $2B21 : Math argument.
11042 : $2B22 : Result is too large.
11043 : $2B23 : File already exists.
11047 : $2B27 : Unknown internal operating system error.
11058 : $2B32 : Share violation.
11059 : $2B33 : Lock violation.
11060 : $2B34 : Critical DOS Error.
11061 : $2B35 : Drive not ready.
11108 : $2B64 : Not exact read/write.
11109 : $2B65 : Operating system network error.
11110 : $2B66 : Error from NOVELL file server.
11111 : $2B67 : NOVELL server out of memory.
11112 : $2B68 : Record already locked by this workstation.
11113 : $2B69 : Record not locked.

Network related

11265 : $2C01 : Network initialization failed.
11266 : $2C02 : Network user limit exceeded.
11267 : $2C03 : Wrong .NET file version.
11268 : $2C04 : Cannot lock network file.
11269 : $2C05 : Directory is not private.
11270 : $2C06 : Directory is controlled by other .NET file.
11271 : $2C07 : Unknown network error.

11272 : $2C08 : Not initialized for accessing network files.
11273 : $2C09 : SHARE not loaded. It is required to share local files.
11274 : $2C0A : Not on a network. Not logged in or wrong network driver.
11275 : $2C0B : Lost communication with SQL server.
11277 : $2C0D : Cannot locate or connect to SQL server.
11278 : $2C0E : Cannot locate or connect to network server.

Optional parameter related

11521 : $2D01 : Optional parameter is required.
11522 : $2D02 : Invalid optional parameter.

Query related

11777 : $2E01 : obsolete
11778 : $2E02 : obsolete
11779 : $2E03 : Ambiguous use of ! (inclusion operator).
11780 : $2E04 : obsolete
11781 : $2E05 : obsolete
11782 : $2E06 : A SET operation cannot be included in its own grouping.
11783 : $2E07 : Only numeric and date/time fields can be averaged.
11784 : $2E08 : Invalid expression.
11785 : $2E09 : Invalid OR expression.
11786 : $2E0A : obsolete
11787 : $2E0B : bitmap
11788 : $2E0C : CALC expression cannot be used in INSERT, DELETE, CHANGETO and SET rows.
11789 : $2E0D : Type error in CALC expression.
11790 : $2E0E : CHANGETO can be used in only one query form at a time.
11791 : $2E0F : Cannot modify CHANGED table.
11792 : $2E10 : A field can contain only one CHANGETO expression.
11793 : $2E11 : A field cannot contain more than one expression to be inserted.
11794 : $2E12 : obsolete
11795 : $2E13 : CHANGETO must be followed by the new value for the field.
11796 : $2E14 : Checkmark or CALC expressions cannot be used in FIND queries.
11797 : $2E15 : Cannot perform operation on CHANGED table together with a CHANGETO query.
11798 : $2E16 : chunk
11799 : $2E17 : More than 255 fields in ANSWER table.
11800 : $2E18 : AS must be followed by the name for the field in the ANSWER table.
11801 : $2E19 : DELETE can be used in only one query form at a time.
11802 : $2E1A : Cannot perform operation on DELETED table together with a DELETE query.
11803 : $2E1B : Cannot delete from the DELETED table.
11804 : $2E1C : Example element is used in two fields with

incompatible types or with a BLOB.
11805 : $2E1D : Cannot use example elements in an OR expression.
11806 : $2E1E : Expression in this field has the wrong type.
11807 : $2E1F : Extra comma found.
11808 : $2E20 : Extra OR found.
11809 : $2E21 : One or more query rows do not contribute to the ANSWER.
11810 : $2E22 : FIND can be used in only one query form at a time.
11811 : $2E23 : FIND cannot be used with the ANSWER table.
11812 : $2E24 : A row with GROUPBY must contain SET operations.
11813 : $2E25 : GROUPBY can be used only in SET rows.
11814 : $2E26 : Use only INSERT, DELETE, SET or FIND in leftmost column.
11815 : $2E27 : Use only one INSERT, DELETE, SET or FIND per line.
11816 : $2E28 : Syntax error in expression.
11817 : $2E29 : INSERT can be used in only one query form at a time.
11818 : $2E2A : Cannot perform operation on INSERTED table together with an INSERT query.
11819 : $2E2B : INSERT, DELETE, CHANGETO and SET rows may not be checked.
11820 : $2E2C : Field must contain an expression to insert (or be blank).
11821 : $2E2D : Cannot insert into the INSERTED table.
11822 : $2E2E : Variable is an array and cannot be accessed.
11823 : $2E2F : Label
11824 : $2E30 : Rows of example elements in CALC expression must be linked.
11825 : $2E31 : Variable name is too long.
11826 : $2E32 : Query may take a long time to process.
11827 : $2E33 : Reserved word or one that can't be used as a variable name.
11828 : $2E34 : Missing comma.
11829 : $2E35 : Missing).
11830 : $2E36 : Missing right quote.
11831 : $2E37 : Cannot specify duplicate column names.
11832 : $2E38 : Query has no checked fields.
11833 : $2E39 : Example element has no defining occurrence.
11834 : $2E3A : No grouping is defined for SET operation.
11835 : $2E3B : Query makes no sense.
11836 : $2E3C : Cannot use patterns in this context.
11837 : $2E3D : Date does not exist.
11838 : $2E3E : Variable has not been assigned a value.
11839 : $2E3F : Invalid use of example element in summary expression.
11840 : $2E40 : Incomplete query statement. Query only contains a SET definition.
11841 : $2E41 : Example element with ! makes no sense in expression.
11842 : $2E42 : Example element cannot be used more than twice with a ! query.
11843 : $2E43 : Row cannot contain expression.

11844 : $2E44 : obsolete
11845 : $2E45 : obsolete
11846 : $2E46 : No permission to insert or delete records.
11847 : $2E47 : No permission to modify field.
11848 : $2E48 : Field not found in table.
11849 : $2E49 : Expecting a column separator in table header.
11850 : $2E4A : Expecting a column separator in table.
11851 : $2E4B : Expecting column name in table.
11852 : $2E4C : Expecting table name.
11853 : $2E4D : Expecting consistent number of columns in all rows of table.
11854 : $2E4E : Cannot open table.
11855 : $2E4F : Field appears more than once in table.
11856 : $2E50 : This DELETE, CHANGE or INSERT query has no ANSWER.
11857 : $2E51 : Query is not prepared. Properties unknown.
11858 : $2E52 : DELETE rows cannot contain quantifier expression.
11859 : $2E53 : Invalid expression in INSERT row.
11860 : $2E54 : Invalid expression in INSERT row.
11861 : $2E55 : Invalid expression in SET definition.
11862 : $2E56 : row use
11863 : $2E57 : SET keyword expected.
11864 : $2E58 : Ambiguous use of example element.
11865 : $2E59 : obsolete
11866 : $2E5A : obsolete
11867 : $2E5B : Only numeric fields can be summed.
11868 : $2E5C : Table is write protected.
11869 : $2E5D : Token not found.
11870 : $2E5E : Cannot use example element with ! more than once in a single row.
11871 : $2E5F : Type mismatch in expression.
11872 : $2E60 : Query appears to ask two unrelated questions.
11873 : $2E61 : Unused SET row.
11874 : $2E62 : INSERT, DELETE, FIND, and SET can be used only in the leftmost column.
11875 : $2E63 : CHANGETO cannot be used with INSERT, DELETE, SET or FIND.
11876 : $2E64 : Expression must be followed by an example element defined in a SET.
11877 : $2E65 : Lock failure.
11878 : $2E66 : Expression is too long.
11879 : $2E67 : Refresh exception during query.
11880 : $2E68 : Query canceled.
11881 : $2E69 : Unexpected Database Engine error.
11882 : $2E6A : Not enough memory to finish operation.
11883 : $2E6B : Unexpected exception.
11884 : $2E6C : Feature not implemented yet in query.
11885 : $2E6D : Query format is not supported.
11886 : $2E6E : Query string is empty.
11887 : $2E6F : Attempted to prepare an empty query.
11888 : $2E70 : Buffer too small to contain query string.
11889 : $2E71 : Query was not previously parsed or prepared.
11890 : $2E72 : Function called with bad query handle.
11891 : $2E73 : QBE syntax error.
11892 : $2E74 : Query extended syntax field count error.

11893 : $2E75 : Field name in sort or field clause not found.
11894 : $2E76 : Table name in sort or field clause not found.
11895 : $2E77 : Operation is not supported on BLOB fields.
11896 : $2E78 : General BLOB error.
11897 : $2E79 : Query must be restarted.
11898 : $2E7A : Unknown answer table type.
11926 : $2E96 : Blob cannot be used as grouping field.
11927 : $2E97 : Query properties have not been fetched.
11928 : $2E98 : Answer table is of unsuitable type.
11929 : $2E99 : Answer table is not yet supported under server alias.
11930 : $2E9A : Non-null blob field required. Can't insert records
11931 : $2E9B : Unique index required to perform changeto
11932 : $2E9C : Unique index required to delete records
11933 : $2E9D : Update of table on the server failed.
11934 : $2E9E : Can't process this query remotely.
11935 : $2E9F : Unexpected end of command.
11936 : $2EA0 : Parameter not set in query string.
11937 : $2EA1 : Query string is too long.
11946 : $2EAA : No such table or correlation name.
11947 : $2EAB : Expression has ambiguous data type.
11948 : $2EAC : Field in order by must be in result set.
11949 : $2EAD : General parsing error.
11950 : $2EAE : Record or field constraint failed.
11951 : $2EAF : When GROUP BY exists, every simple field in projectors must be in GROUP BY.
11952 : $2EB0 : User defined function is not defined.
11953 : $2EB1 : Unknown error from User defined function.
11954 : $2EB2 : Single row subquery produced more than one row.
11955 : $2EB3 : Expressions in group by are not supported.
11956 : $2EB4 : Queries on text or ascii tables is not supported.
11957 : $2EB5 : ANSI join keywords USING and NATURAL are not supported in this release.
11958 : $2EB6 : SELECT DISTINCT may not be used with UNION unless UNION ALL is used.
11959 : $2EB7 : GROUP BY is required when both aggregate and non-aggregate fields are used in result set.
11960 : $2EB8 : INSERT and UPDATE operations are not supported on autoincrement field type.
11961 : $2EB9 : UPDATE on Primary Key of a Master Table may modify more than one record.
11962 : $2EBA : Queries on MS ACCESS tables are not supported by local query engines.
11963 : $2EBB : Preparation of field-level constraint failed.
11964 : $2EBC : Preparation of field default failed.
11965 : $2EBD : Preparation of record-level constraint failed.
11972 : $2EC4 : Constraint Failed. Expression:

Version Mismatch Category

12033 : $2F01 : Interface mismatch. Engine version different.
12034 : $2F02 : Index is out of date.
12035 : $2F03 : Older version (see context).
12036 : $2F04 : .VAL file is out of date.
12037 : $2F05 : BLOB file version is too old.

12038 : $2F06 : Query and Engine DLLs are mismatched.
12039 : $2F07 : Server is incompatible version.
12040 : $2F08 : Higher table level required

Capability not supported

12289 : $3001 : Capability not supported.
12290 : $3002 : Not implemented yet.
12291 : $3003 : SQL replicas not supported.
12292 : $3004 : Non-blob column in table required to perform operation.
12293 : $3005 : Multiple connections not supported.
12294 : $3006 : Full dBASE expressions not supported.

System configuration error

12545 : $3101 : Invalid database alias specification.
12546 : $3102 : Unknown database type.
12547 : $3103 : Corrupt system configuration file.
12548 : $3104 : Network type unknown.
12549 : $3105 : Not on the network.
12550 : $3106 : Invalid configuration parameter.

Warnings

12801 : $3201 : Object implicitly dropped.
12802 : $3202 : Object may be truncated.
12803 : $3203 : Object implicitly modified.
12804 : $3204 : Should field constraints be checked?
12805 : $3205 : Validity check field modified.
12806 : $3206 : Table level changed.
12807 : $3207 : Copy linked tables?
12809 : $3209 : Object implicitly truncated.
12810 : $320A : Validity check will not be enforced.
12811 : $320B : Multiple records found, but only one was expected.
12812 : $320C : Field will be trimmed, cannot put master records into PROBLEM table.

Miscellaneous

13057 : $3301 : File already exists.
13058 : $3302 : BLOB has been modified.
13059 : $3303 : General SQL error.
13060 : $3304 : Table already exists.
13061 : $3305 : Paradox 1.0 tables are not supported.
13062 : $3306 : Update aborted.

Compatibility related

13313 : $3401 : Different sort order.
13314 : $3402 : Directory in use by earlier version of Paradox.
13315 : $3403 : Needs Paradox 3.5-compatible language driver.

Data Repository related

13569 : $3501 : Data Dictionary is corrupt
13570 : $3502 : Data Dictionary Info Blob corrupted
13571 : $3503 : Data Dictionary Schema is corrupt
13572 : $3504 : Attribute Type exists
13573 : $3505 : Invalid Object Type
13574 : $3506 : Invalid Relation Type
13575 : $3507 : View already exists
13576 : $3508 : No such View exists
13577 : $3509 : Invalid Record Constraint
13578 : $350A : Object is in a Logical DB
13579 : $350B : Dictionary already exists
13580 : $350C : Dictionary does not exist
13581 : $350D : Dictionary database does not exist
13582 : $350E : Dictionary info is out of date - needs Refresh
13584 : $3510 : Invalid Dictionary Name
13585 : $3511 : Dependent Objects exist
13586 : $3512 : Too many Relationships for this Object Type
13587 : $3513 : Relationships to the Object exist
13588 : $3514 : Dictionary Exchange File is corrupt
13589 : $3515 : Dictionary Exchange File Version mismatch
13590 : $3516 : Dictionary Object Type Mismatch
13591 : $3517 : Object exists in Target Dictionary
13592 : $3518 : Cannot access Data Dictionary
13593 : $3519 : Cannot create Data Dictionary
13594 : $351A : Cannot open Database

Driver related

15873 : $3E01 : Wrong driver name.
15874 : $3E02 : Wrong system version.
15875 : $3E03 : Wrong driver version.
15876 : $3E04 : Wrong driver type.
15877 : $3E05 : Cannot load driver.
15878 : $3E06 : Cannot load language driver.
15879 : $3E07 : Vendor initialization failed.
15880 : $3E08 : Your application is not enabled for use with
this driver.

Delphi 3 file types with descriptions

NUMBER : 3213
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : July 7, 1997

TITLE : Delphi 3 file types with descriptions

The .CAB File Format

This is the file format that Delphi now offers its users for web deployment. The cabinet format is an efficient way to package multiple files. The cabinet format has two key features: multiple files can be stored in a single cabinet (.cab file) and data compression is performed across file boundaries, which significantly improves the compression ratio. Cabinet file construction can be designed around the number of files to be compressed and the expected patterns for gaining access to them (sequential, random, all at once, or a few at a time). Delphi does not take advantage file compression across file boudaries.

The .LIC File Format

There really is no .lic file format per se. These files are generally just text files that contain a key string or two.

The .INF File Format

All inf files are made up of sections and items. Each named section contains its associated items. All inf files start with the header section. After the header, sections can be laid out in any order desired. Each header is as follows [HeaderName]. This is followed by the items: ItemA = ItemDetail. For detailed information on this topic please see the Device Information File Reference.

The .dpr file format.

The .dpr file is the central file to a delphi project. It serves as the primary entry point for the executable. The dpr contains the references to the other files in the project and links forms with their associated units. This file should be edited with care as changes in it can prevent your project from loading. This file is critical for loading or moving(copying) the project.

The .pas file format.

This is a standard text file that can be edited in a text editor. Edit this file carfully as it may result in the loss of some advantages of the two way tool. For example pasting code for a button into the type declaration for a form does not result in the corresponding entry into the .dfm file. All pas files in a project are critical for rebuilding the project.

The .dfm file format.

This file contains the details of the objects contained in a form. It can be view as text by right clicking on the form and selecting view as text from the pop-up menu, or it can be converted to text and back using the convert.exe found in the bin directory. Caution should be used in altering this file as changes to it could prevent the IDE from being able to load the form. This file is critical to moving or rebuilding the project.

The .DOF File Format

This text file contains the current settings for project options, such as compiler and linker settings, directories, conditional directives, and command-line parameters. These settings can be customized on a project-by-project basis.

The .DSK File Format

This text file stores information about the state of your project, such as which windows are open and what position they are in. Like the .DOF file, this file can be customized project-by-project.

The .DPK File Format

This file contains the source code for a package (analogous to the .DPR in a standard Delphi project) Like the .DPR file the .DPK file is a plain text file that can be edited (with caution) using a standard editor. One of the primary reasons you may need to do this would be if you were using the command line compiler.

The .DCP File Format

This binary image file consists of the actual compiled package. Symbol information and additional header information required by the IDE are all contained within the .DCP file. The IDE must have access to this file in order to build a project.

The .DPL File Format

This is the actual executable runtime package. This file is a Windows DLL with Delphi-specific features integrated into it. This file is essential for deployment of an application that uses a package.

The .DCI File Format
This text file contains both standard and user-defined code templates for use within the IDE. The file can be edited with a standard text editor or through the IDE. As with any text data file used by Delphi, modifying the file directly is discouraged.

The .DCT File Format

This proprietary binary file contains the user-defined component template information. This file is not meant to be edited by any means other than through the IDE. Since this file is proprietary the format and comptability with future versions of Delphi may likely change.

The .TLB File Format

The .TLB file is a proprietary binary type library file. This file provides a way for identifying what types of objects and interfaces are available on an ActiveX server. Like a unit or a header file the .TLB serves as a repository for necessary symbol information for an application. Since this file is proprietary the format and comptability with future versions of Delphi may likely change.

The .DRO File Format

This text file contains information about the object repository. Each entry in this file contains specific information about each available item in the object repository. Even though this file is a standard text file it is not recommended that you edit it by hand. The repository should only be modified from the Tools|Repository menu in the IDE.

The .RES File Format

This standard binary windows-format resource file includes information about an application. By default Delphi creates a new .RES file every time a project is compiled into an application.

The .DB File Format

Files with this extension are standard Paradox files.

The .DBF File Format

Files with this extension are standard dBASE files.

The .GDB File Format

Files with this extension are standard Interbase files.

The .DMT File Format

This proprietary binary file contains the shipped and user-defined menu templates information. This file is not meant to be edited by any means other

than through the IDE. Since this file is proprietary the format and comptability with future versions of Delphi may likely change.

The .DBI File Format

This text file contains initialization information for the Database Explorer. This file is not meant to be edited by any means other than through the Database Explorer.

The .DEM File Format

This text file contains some standard country-specific formats for a TMaskEdit component. As with any text data file used by Delphi, modifying the file directly is discouraged.

The .OCX File Format

An .OCX file is a specialized DLL which contains all or some associated functions of an ActiveX control. The OCX file can be thought of as a "wrapper" which contains the object and it's means of communications with other objects and servers.

Direct Commands to Printer - Passthrough/Escape

NUMBER : 3196
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : February 28, 1997

TITLE : Direct Commands to Printer - Passthrough/Escape

Although Delphi's TPrinter unit makes it easy to interface to a given printer, there are times when you may need to drop down to the printers level and send device specific escape sequences. Under 16-bit versions of Windows, this was as easy as opening the printer port, but under Windows NT, directly accessing the hardware is is illegal. One solution is to use the Windows "PASSTHROUGH" escape to send an escape sequence directly to the printer. In order to use the "PASSTHROUGH" escape, it must be supported by the printer driver. Be forwarned that not all printer drivers will support this feature.

It's worth noting that the "PASSTHROUGH" escape is documented as obsolete for thirty-two bit applications. It should be a number of years before this escape goes by the way, since it is used in many commercial applications.

The example code presented is not targeted to any specific printer model. You will need to know the correct escape sequences to send to the printer you are interfacing to. Note that you must still call the BeginDoc and EndDoc methods of TPrinter. During the BeginDoc call, the printer driver initializes the printer as necessary, and during the EndDoc call, the printer driver will uninitialize the printer and eject the page. When you do make your escape call, the printer may be set for the current windows mapping mode if the printer supports scaling internaly. Technically, you should not do anything that would cause the printer memory to be reset, or eject a page with an escape sequence. In other words, try to leave the printer in the same state it was in when you made your escape. This is more important on intellegent printers (Postscript printers), and not important at all on a standard TTY line printer, where you can do just about anything you like, including ejecting pages.

Code Example:

You will need to declare a structure to hold the buffer you are sending. The structure of the buffer is defined as a word containing the length of the buffer, followed by the buffer containing the data.

Before making the escape call to pass the data, we will use the escape "QUERYESCSUPPORT" to determine if the "PASSTHROUGH" escape is supported by the print driver.

Finally, be aware that your data will be inserted directly into
the printers data stream. On some printer models (Postscript),
you may need to add a space to the start and end of your data
to separate your data from the printer drivers data.

(Postscript is a Registered Trademark of Adobe Systems Incorporated)

```
unit Esc1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{ add the printers unit }
uses
   Printers;

{$R *.DFM}

{ declare the "PASSTHROUGH" structure }
type TPrnBuffRec = record
  BuffLength : word;
  Buffer : array [0..255] of char;
end;


procedure TForm1.Button1Click(Sender: TObject);
var
  Buff : TPrnBuffRec;
  TestInt : integer;
  s : string;
begin

{ Test to see if the "PASSTHROUGH" escape is supported }
  TestInt := PASSTHROUGH;
  if Escape(Printer.Handle,
            QUERYESCSUPPORT,
            sizeof(TestInt),
```

```
                @TestInt,
                nil) > 0 then begin

  { Start the printout }
    Printer.BeginDoc;

  { Make a string to passthrough }
    s := ' A Test String ';

  { Copy the string to the buffer }
    StrPCopy(Buff.Buffer, s);

  { Set the buffer length }
    Buff.BuffLength := StrLen(Buff.Buffer);

  { Make the escape}
    Escape(Printer.Canvas.Handle,
           PASSTHROUGH,
           0,
           @Buff,
           nil);

  { End the printout }
    Printer.EndDoc;
  end;
end;

end.
```

An example of drag and drop between DBGrids

NUMBER : 3215
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : July 7, 1997

TITLE : An example of drag and drop between DBGrids

Title: An example of drag and drop between DBGrids

This sample component and sample project demonstrates an easy way of enabling drag and drop of an arbitrary field in one data aware grid onto an arbitrary field in another data aware grid.

1. Launch Delphi 3 (the code will work in 1 and 2 as well with some minor changes).

2. Do a File|New|Unit. Take the MyDBGrid unit (below) and paste it in the newly created unit. Do a File|Save As. Save the unit as MyDBGrid.pas.

3. Do a Component|Install Component. Switch to the Info New Package tab. Put MyDBGrid.pas in the Unit file name box. Call the package MyPackage.dpk. Hit Yes when Delphi 3 tells you that the package will be built and installed. Hit OK when Delphi 3 tells you that VCL30.DPL is needed. The package will now be rebuilt and installed. You will now find the TMyDBGrid component on your Samples tab on your component palette. Close the package editor and save the package.

4. Do a File|New Application. Right click on the form (Form1) and select View As Text. Take the GridU1 form source (below) and paste it in Form1. Right click on the form and select View As Form. This may take a few moments since it's opening up the tables for you. Take the GridU1 unit (below) and paste it in the unit (Unit1).

5. Do a File|Save Project As. Save the unit as GridU1.pas. Save the project as GridProj.dpr.

6. Now, run the project and enjoy the dragging and dropping of fields inbetween or with the two grids.

The MyDBGrid unit

```
unit MyDBGrid;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids;
```

```
type
  TMyDBGrid = class(TDBGrid)
  private
    { Private declarations }
    FOnMouseDown: TMouseEvent;
  protected
    { Protected declarations }
    procedure MouseDown(Button: TMouseButton; Shift: TShiftState;
      X, Y: Integer); override;
  published
    { Published declarations }
    property Row;
    property OnMouseDown read FOnMouseDown write FOnMouseDown;
  end;

procedure Register;

implementation

procedure TMyDBGrid.MouseDown(Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Assigned(FOnMouseDown) then
    FOnMouseDown(Self, Button, Shift, X, Y);
  inherited MouseDown(Button, Shift, X, Y);
end;

procedure Register;
begin
  RegisterComponents('Samples', [TMyDBGrid]);
end;

end.
```

The GridU1 unit

```
unit GridU1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Db, DBTables, Grids, DBGrids, MyDBGrid, StdCtrls;

type
  TForm1 = class(TForm)
    MyDBGrid1: TMyDBGrid;
    Table1: TTable;
    DataSource1: TDataSource;
    Table2: TTable;
    DataSource2: TDataSource;
    MyDBGrid2: TMyDBGrid;
    procedure MyDBGrid1MouseDown(Sender: TObject;
```

```
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure MyDBGrid1DragOver(Sender, Source: TObject;
      X, Y: Integer; State: TDragState; var Accept: Boolean);
    procedure MyDBGrid1DragDrop(Sender, Source: TObject;
      X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

var
  SGC : TGridCoord;

procedure TForm1.MyDBGrid1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  DG : TMyDBGrid;
begin
  DG := Sender as TMyDBGrid;
  SGC := DG.MouseCoord(X,Y);
  if (SGC.X > 0) and (SGC.Y > 0) then
    (Sender as TMyDBGrid).BeginDrag(False);
end;

procedure TForm1.MyDBGrid1DragOver(Sender, Source: TObject;
  X, Y: Integer; State: TDragState; var Accept: Boolean);
var
  GC : TGridCoord;
begin
  GC := (Sender as TMyDBGrid).MouseCoord(X,Y);
  Accept := Source is TMyDBGrid and (GC.X > 0) and (GC.Y > 0);
end;

procedure TForm1.MyDBGrid1DragDrop(Sender, Source: TObject;
  X, Y: Integer);
var
  DG       : TMyDBGrid;
  GC       : TGridCoord;
  CurRow : Integer;
begin
  DG := Sender as TMyDBGrid;
  GC := DG.MouseCoord(X,Y);
  with DG.DataSource.DataSet do begin
    with (Source as TMyDBGrid).DataSource.DataSet do
      Caption := 'You dragged '"'+Fields[SGC.X-1].AsString+'"';
    DisableControls;
    CurRow := DG.Row;
    MoveBy(GC.Y-CurRow);
```

```
      Caption := Caption+' to "'+Fields[GC.X-1].AsString+'"';
      MoveBy(CurRow-GC.Y);
      EnableControls;
    end;
end;

end.
```

The GridU1 form

```
object Form1: TForm1
  Left = 200
  Top = 108
  Width = 544
  Height = 437
  Caption = 'Form1'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 13
  object MyDBGrid1: TMyDBGrid
    Left = 8
    Top = 8
    Width = 521
    Height = 193
    DataSource = DataSource1
    Row = 1
    TabOrder = 0
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    OnDragDrop = MyDBGrid1DragDrop
    OnDragOver = MyDBGrid1DragOver
    OnMouseDown = MyDBGrid1MouseDown
  end
  object MyDBGrid2: TMyDBGrid
    Left = 7
    Top = 208
    Width = 521
    Height = 193
    DataSource = DataSource2
    Row = 1
    TabOrder = 1
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    OnDragDrop = MyDBGrid1DragDrop
```

```
    OnDragOver = MyDBGrid1DragOver
    OnMouseDown = MyDBGrid1MouseDown
  end
  object Table1: TTable
    Active = True
    DatabaseName = 'DBDEMOS'
    TableName = 'ORDERS'
    Left = 104
    Top = 48
  end
  object DataSource1: TDataSource
    DataSet = Table1
    Left = 136
    Top = 48
  end
  object Table2: TTable
    Active = True
    DatabaseName = 'DBDEMOS'
    TableName = 'CUSTOMER'
    Left = 104
    Top = 240
  end
  object DataSource2: TDataSource
    DataSet = Table2
    Left = 136
    Top = 240
  end
end
```

Looping Through the Controls and Components Arrays

NUMBER : 3216
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : July 7, 1997

TITLE : Looping Through the Controls and Components Arrays

There are 2 important properties in Delphi that everyone using the product should know about. The components array is a property of all components that own other components. The controls array is a property of all TWinControls that are parents of TWinControls. These properties are so important because they allow you to iterate through a group of components without knowing the names or knowing what type of controls or components they are. This could be important for example if you wanted to disable or enable all the controls on a form or within a container control (such as a panel). Proper code for this iterative process is shown below for both cases.

1. Iteration through the components array of the form:

```
begin
  for i:= 0 to componentcount - 1 do
  begin
    {enter the code to act on each member of the array here}
  end;
end;
```

2. Iteration through the controls array of the form:

```
begin
  for i:= 0 to controlcount - 1 do
  begin
    {enter the code to act on each member of the array here}
  end;
end;
```

The variable 'i' is an integer declared for iteration one array member at a time and must be declared by the user. The variables controlcount and componentcount are additional properties of TWinControl and TComponent respectively. These properties are often used in conjunction with iteration through the corresponding array properties.

An Overview of Borland Online Information Services

NUMBER : 9604
PRODUCT : Borland
VERSION : All
OS : All
DATE : April 25, 1996

TITLE : An Overview of Borland Online Information Services

Any time of any day, you can use your modem and a communications program to gain immediate access to a vast source of information and files related to Borland products. Your call to the Borland Download Bulletin Board System (DLBBS) will never be put on hold. In a matter of minutes, you receive the sample programs, macros, drivers, or utilities you need - transferred directly to your computer, so you never have to type them in by hand. Find answers to your questions 24 hours per day by accessing our frequently updated collection of technical information files containing tips on how to get the most out of Borland products. There is no charge to use the Borland Download Bulletin Board system.

The Borland areas on the CompuServe Information Service (CIS), the Byte Information eXchange (BIX), and the General Electric Network for Information Exchange (GEnie) feature the same benefits as the Borland Download Bulletin Board System (DLBBS), plus the added benefit of electronic messages. Charges for using CompuServe, BIX and GEnie vary.

	Voice Contact	Pay/Free	Files	Live "chat"	Messages	24hrs
DLBBS		Free	Yes	No	No	Yes
CIS	800/848-8199 ask for Rep 62	Pay	Yes	Yes	Yes	Yes
BIX	800/227-2983	Pay	Yes	Yes	Yes	Yes
GEnie	800/638-9636	Pay	Yes	Yes	Yes	Yes

HOW TO LOG ON:

For Borland's DLBBS, set your communications program to dial 408/431-5096 using the following communications parameters: 9600 baud, 2400 baud or 1200 baud, 8 data bits, no parity, one stop bit. After a connection is established, press <Enter>.
For CIS, BIX, and GEnie, you may establish an ID and receive logon instructions by calling the appropriate phone number shown above.

To access a Borland area on:	Type:
CompuServe (CIS)	GO BORLAND
BIX	JOIN BORLAND
GEnie	BORLAND

Technical Support via The Internet:

Users of all Borland products now have the option to retrieve technical resources and information via 'anonymous' File Transfer Protocol (FTP) over the Internet. FTP is the Internet standard for transferring files to and from a remote network host. For more information about receiving Technical Support via the Internet refer to Technical Information Sheet #9656.

Note: The Borland FTP site is also accessible via the Tech Info pages for each product on www.borland.com. From www.borland.com, choose Tech Info, then choose the product. On each Tech Info page there is a link to Technical Information. This is the easiest way to find technical information on the Borland FTP site.

Displaying System Resources in Win 95 and NT 4

NUMBER : 3231
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : July 22, 1997

TITLE : Displaying System Resources in Win 95 and NT 4

Displaying System Resources

This document describes the displaying of total system resources or the amount of available system resources for Windows 95 or Windows NT 4.0. It includes the structure of the TMemoryStatus structure, the function GlobalMemoryStatus that populates this structure as defined in Windows.pas, and the Windows API header description. It also includes a code example of populating a memo with this information in Delphi.

TMemoryStatus source can be found in Windows.PAS which is in the Source\RTL\WIN directory if you have the Client\Server Suite of Delphi.

Structure of TMemoryStatus:

```
TMemoryStatus = record
  dwLength: DWORD;
  dwMemoryLoad: DWORD;
  dwTotalPhys: DWORD;
  dwAvailPhys: DWORD;
  dwTotalPageFile: DWORD;
  dwAvailPageFile: DWORD;
  dwTotalVirtual: DWORD;
  dwAvailVirtual: DWORD;
```

Function called to populate TMemoryStatus:

```
procedure GlobalMemoryStatus(var lpBuffer: TMemoryStatus); stdcall;
```

WINAPI help for said function:

```
VOID GlobalMemoryStatus(
  // pointer to the memory status structure
  LPMEMORYSTATUS  lpBuffer
);
```

Code for populating a TMemo with Information about system resources:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
```

```
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  MemoryStatus: TMemoryStatus;

begin

  Memo1.Lines.Clear;

  MemoryStatus.dwLength := SizeOf(MemoryStatus);

  GlobalMemoryStatus(MemoryStatus);

  with MemoryStatus do
  begin
// Size of MemoryStatus record
    Memo1.Lines.Add(IntToStr(dwLength) +
      ' Size of "MemoryStatus" record');
// Per-Cent of Memory in use by your system
    Memo1.Lines.Add(IntToStr(dwMemoryLoad) +
      '% memory in use');
// The amount of Total Physical memory allocated to your system.
    Memo1.Lines.Add(IntToStr(dwTotalPhys) +
      ' Total Physical Memory in bytes');
// The amount available of physical memory in your system.
    Memo1.Lines.Add(IntToStr(dwAvailPhys) +
      ' Available Physical Memory in bytes');
// The amount of Total Bytes allocated to your page file.
    Memo1.Lines.Add(IntToStr(dwTotalPageFile) +
      ' Total Bytes of Paging File');
// The amount of available bytes in your page file.
    Memo1.Lines.Add(IntToStr(dwAvailPageFile) +
      ' Available bytes in paging file');
// The amount of Total bytes allocated to this program
// (generally 2 gigabytes of virtual space).
    Memo1.Lines.Add(IntToStr(dwTotalVirtual) +
      ' User Bytes of Address space');
```

```
// The amount of avalable bytes that is left to your program to use.
      Memo1.Lines.Add(IntToStr(dwAvailVirtual) +
        ' Available User bytes of address space');
   end; // with
end; // procedure

end.
```

Sample Output of what is contained in Memo1.Lines:

```
32: Size of 'MemoryStatus' record in bytes
76%: memory in use
33054720: Total Physical Memory in bytes
499712: Available Physical Memory in bytes
53608448: Total Bytes of Paging File
36372480: Available bytes in paging file
2143289344: User Bytes of Address space
2135556096: Available User bytes of address space
```

Edit Controls that Align Under NT 4

NUMBER : 3232
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : July 22, 1997

TITLE : Edit Controls that Align Under NT 4

Creating an edit control that allows alignment under NT4.

OVERVIEW

This document demonstrates how to create a descendant of TEdit that allows alignment of the text.

WHO SHOULD USE THIS DOCUMENT

Anyone with a basic familiarity with Delphi programming. Applies to any version of Delphi.

CREATING JUSTAEDIT

One of the features the TMemo surfaces that a TEdit does not is the ability to justify the text. This functionality exists via the Alignment property. Great! So why doesn't a TEdit have this functionality, or perhaps a more pertinent question is how do we add it?

This property, unfortunately can not be surfaced from an ancestor, because the functionality does not exist in any ancestor. So it needs to be implemented via the window style flags. ES_LEFT, ES_RIGHT, and ES_CENTER specify the text alingment for edit controls. These flags do not have any bearing on single line edit controls...unless running NT 4, Service pack 3, and that is the reason the functionality does not exist in the native control. This functionality in now implemented in NT 4.0 but will have no effect in Win 3.1 and Windows 95.

WORKING CODE

```
unit JusEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
 Dialogs,
  StdCtrls;

type
  TJustaEdit = class(TEdit)
  private
```

```
    { Private declarations }
     fAlignment : TAlignment;
  protected
    { Protected declarations }
    procedure SetAlignment(Value: TAlignment);
  public
    { Public declarations }
     procedure createParams(var Params : TCreateParams); override;
  published
    { Published declarations }
     property Alignment: TAlignment read FAlignment write SetAlignment
      default taLeftJustify;
  end;

procedure Register;

implementation
procedure TJustaEdit.CreateParams(var Params : TCreateParams);
var
  x : Longint;
begin
  inherited CreateParams(Params);
  case fAlignment of
    tarightjustify: x := es_right;
    taleftjustify : x := es_left;
    tacenter      : x := es_center;
  end;
  params.style := params.style or x;


end;
procedure TJustaEdit.SetAlignment;
begin
   if FAlignment <> Value then
  begin
    FAlignment := Value;
    RecreateWnd;
  end;
end;
procedure Register;
begin
  RegisterComponents('Samples', [TJustaEdit]);
end;

end.
```

How do I map a network drive in Windows NT or '95?

NUMBER : 3233
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : July 22, 1997

TITLE : How do I map a network drive in Windows NT or '95?

How do I map a network drive in Windows NT or '95?

You can use the WNetAddConnection2 API call. The prototype for the API call is in Windows.Pas.

```
function WNetAddConnection2W(var lpNetResource: TNetResourceW;
  lpPassword, lpUserName: PWideChar;
  dwFlags: DWORD): DWORD; stdcall;
```

To make the call you will need to fill a lpNetResource structure with a minimum set of parameters, shown in the example below. You pass this structure as the first parameter to the call, the password, user name, and a flag that indicates whether this mapping should be persistant every time the machine is logged onto. For more info on the API itself, see Window's Programmers Reference help (find the function in Windows.pas, place your text cursor over the function call, and hit F1 to bring up help).

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NRW: TNetResource;
begin
  with NRW do
  begin
    dwType := RESOURCETYPE_ANY;
    lpLocalName := 'X:'; // map to this driver letter
    lpRemoteName := '\\MyServer\MyDirectory';
    // Must be filled in. If an empty string is used,
    // it will use the lpRemoteName.
    lpProvider := '';
  end;
  WNetAddConnection2(NRW, 'MyPassword', 'MyUserName',
    CONNECT_UPDATE_PROFILE);
end;
```

Adding shortcuts to Win95/WinNT4 Desktop/StartMenu

NUMBER : 3234
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : July 22, 1997

TITLE : Adding shortcuts to Win95/WinNT4 Desktop/StartMenu

Title: Adding shortcuts to the Win95/WinNT40 desktop or start menu

This sample project demonstrates an easy way to add shortcuts to
your Windows 95 or Windows NT 4.0 desktop or start menu.

1. Launch Delphi 3.

2. In a new project, drop a TButton on the form (make sure it's
called Button1). Then double click on Button1. Now you can go
ahead and directly replace the code for Unit1 with the code for
Unit1 below.

The program will set up a shortcut either (see the code) on the
desktop or on the start menu. The shortcut will be called FooBar
and it will open up your AUTOEXEC.BAT in NOTEPAD when executed.

It will read the value of the "Desktop" and "Start Menu" strings
from the registry key named (under HKEY_CURRENT_USER):

Software\MicroSoft\Windows\CurrentVersion\Explorer\Shell Folders

---
The Unit1 unit

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}

uses
  ShlObj, ActiveX, ComObj, Registry;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyObject   : IUnknown;
  MySLink    : IShellLink;
  MyPFile    : IPersistFile;
  FileName   : String;
  Directory : String;
  WFileName : WideString;
  MyReg       : TRegIniFile;
begin
  MyObject := CreateComObject(CLSID_ShellLink);
  MySLink := MyObject as IShellLink;
  MyPFile := MyObject as IPersistFile;
  FileName := 'NOTEPAD.EXE';
  with MySLink do begin
    SetArguments('C:\AUTOEXEC.BAT');
    SetPath(PChar(FileName));
    SetWorkingDirectory(PChar(ExtractFilePath(FileName)));
  end;
  MyReg := TRegIniFile.Create(
    'Software\MicroSoft\Windows\CurrentVersion\Explorer');

// Use the next line of code to put the shortcut on your desktop
  Directory := MyReg.ReadString('Shell Folders','Desktop','');

// Use the next three lines to put the shortcut on your start menu
//   Directory := MyReg.ReadString('Shell Folders','Start Menu','')+
//        '\Whoa!';
//   CreateDir(Directory);

  WFileName := Directory+'\FooBar.lnk';
  MyPFile.Save(PWChar(WFileName),False);
  MyReg.Free;
end;

end.
```

Minimizing Application When a Form Minimizes

NUMBER : 3235
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : July 22, 1997

TITLE : Minimizing Application When a Form Minimizes

How do I minimize the application when I minimize a secondary form? I want to replicate the behavior that occurs when I minimize the main form.

You can use the same logic found in \source\vcl\forms.pas that TCustomForm (in Delphi 3) uses when the form receives a system message to minimize. The form traps WM_SYSCOMMAND messages, checks to see if the command is to minimize the form, checks that the form is the MainForm, then performs the application object's Minimize method. If the message doesn't specify SC_MINIMIZE then we call inherited so that default processing can occur. Note: failure to include the call to inherited will probably cause bad things to happen.

```
.
.
procedure WMSysCommand(var Message: TWMSysCommand);
  message WM_SYSCOMMAND;
.
.
procedure TCustomForm.WMSysCommand(var Message: TWMSysCommand);
begin
  if (Message.CmdType and $FFF0 = SC_MINIMIZE) and
    (Application.MainForm = Self) then
    Application.Minimize
  else
    inherited;
end;
```

To get this same behavior you can move this code to your form's unit, and remove the logic that checks that the form is a MainForm.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
```

```
    Button2: TButton;
  private
    procedure WMSysCommand(var Message: TWMSysCommand);
        message WM_SYSCOMMAND;
  end;

var
  Form1: TForm1;

implementation

uses Unit2;

{$R *.DFM}


procedure TForm1.WMSysCommand(var Message: TWMSysCommand);
begin
 if (Message.CmdType and $FFF0 = SC_MINIMIZE) then
    Application.Minimize
  else
    inherited;
end;

end.
```

Graying Out Enabled/Disabled Data Aware Controls

NUMBER : 3239
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : August 11, 1997

TITLE : Graying Out Enabled/Disabled Data Aware Controls

Graying out enabled/disabled data aware controls

Most data aware components are capable of visually showing that they are disabled (by changing the text color to gray) or enabled (by setting the color to a user-defined windows text color). Some data aware controls such as TDBGrid, TDBRichEdit (in Delphi 3.0) and also TDBEdit (when connected to a numeric or date field) do not display this behavior.

The code below uses RTTI (Run Time Type Information) to extract property information and use that information to set the font color to gray if the control is disabled. If the control is enabled, the text color is set to the standard windows text color.

What follows is the step by step creation of a simple example which consists of a TForm with a TButton and a TDBRichEdit that demonstrates this behavior.

1. Select File|New Application from the Delphi menu bar.
2. Drop a TDataSource, a TTable, a TButton and a TDBEdit onto the form.
3. Set the DatabaseName property of the table to 'DBDEMOS'.
4. Set the TableName property of the table to 'ORDERS.DB'.
5. Set the DataSet property of the datasource to 'Table1'.
6. Set the DataSource property of the DBEdit to 'DataSource1'.
7. Set the DataField property of the DBEdit to 'CustNo'.
8. Set the Active property of the DBEdit to 'False'.
9. Add 'TypInfo' to the uses clause of the form.

Below is the actual procedure to put in the implementation section of your unit:

```
// This procedure will either set the text color of a
// dataware control to gray or the user defined color
// constant in clInfoText.

procedure SetDBControlColor(aControl: TControl);
 var
   FontPropInfo: PPropInfo;
 begin
// Check to see if the control is a dataware control
    if (GetPropInfo(aControl.ClassInfo, 'DataSource') = nil) then exit
    else
       begin
// Extract the front property
          FontPropInfo:= GetPropInfo(aControl.ClassInfo, 'Font');
```

```
// Check if the control is enabled/disabled
          if (aControl.Enabled = false) then
// If disabled, set the font color to gray
                TFont(GetOrdProp(aControl, FontPropInfo)).Color:= clGrayText
           else
// If enabled, set the font color to clInfoText
                TFont(GetOrdProp(aControl, FontPropInfo)).Color:= clInfoText;
         end;
 end;
```

The code for the buttonclick event handler should contain:

```
//   This code will cycle through the Controls array and call
//   SetDbControlColor for each control on your form
//   making sure the font text color is set to what it
//   should be.

procedure TForm1.Button1Click(Sender: TObject);
 var
   i: integer;
begin
// Loop through the control array
   for i:= 0 to ControlCount-1 do
        SetDBControlColor(Controls[i]);
end;
```

Exposing a multi string object in COM

NUMBER : 3240
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : August 11, 1997

TITLE : Exposing a multi string object in COM

This document is intended for those using OLE/COM and want to expose a multi string object similar to Delphi's TStrings object. The IStrings interface for COM parallels the functionality of Delphi's TStrings object. For example, in a case where you have an automation client that contains a TStrings descendant, and you would like the automation server to process the strings, the automation server can employ an implementation of an IStrings interface to store and manipulate these strings.

The example project below contains the elements and steps needed to implement an IStrings interface in a COM automation server, and to interact with the interface from a COM automation client. You'll find the complete unit source listings next, then finally a discussion of some of the design choices made in this project.

The example project contains these units:

Project1_TLB: A Pascal wrapper for the type library containing the interface definition.

Unit1: The interface implementation containing storage for interface properties and code to implement interface methods.

Unit2: The main form for the automation server. This unit is not strictly necessary, but provides feedback to know when methods have been successfully called

AutCli: The automation client that obtains a reference to the interface and uses the interface methods.

The general steps involved are listed below. You can compare each of these steps with the unit code that follows.

1) Create a type library and add an interface called IStr, with a single property called Items of type IStrings. See Developer's Guide, chapter 42 for more information on working with type libraries and creating interfaces in the type library editor.

2) In the previous step, adding an interface using the type library editor will cause a Pascal wrapper unit to be produced (in this example, the unit is called Unit1). Unit1 will contain shell implementations of get and set methods of the Items property. In this step you add implementation code to make the get and set

methods functional. Also you need to add a method to create storage for your Items property, and to clean up that storage when the interface is no longer needed. Unit1 uses Unit2. Unit2 contains a form, memo and status bar to display status of each implemtation method, for diagnostic purposes.

3) Create Unit2 containing a form with TMemo and TStatusBar. The form is used to reflect activity in Unit1.pas. Though this step is not strictly necessary, it helps visualize what's happening between the automation client and server.

4) Create an automation client. In this case the unit is called AutCli and contains a memo component and two buttons that assign the Memo's TStrings data to the IStr interface's Items property.

```
{----------------------------------------------------------------------}
unit Project1_TLB;

{ This file contains pascal declarations imported from a type library.
  This file will be written during each import or refresh of the type
  library editor. Changes to this file will be discarded during the
  refresh process. }

{ Project1 Library }
{ Version 1.0 }

interface

uses Windows, ActiveX, Classes, Graphics, OleCtrls, StdVCL;

const
  LIBID_Project1: TGUID = '{E6F9F3B6-FD3C-11D0-908F-00C04FC291A4}';

const

{ Component class GUIDs }
  Class_IStr: TGUID = '{E6F9F3B8-FD3C-11D0-908F-00C04FC291A4}';

type

{ Forward declarations: Interfaces }
  IIStr = interface;
  IIStrDisp = dispinterface;

{ Forward declarations: CoClasses }
  IStr = IIStr;

{ Dispatch interface for IStr Object }

  IIStr = interface(IDispatch)
    ['{E6F9F3B7-FD3C-11D0-908F-00C04FC291A4}']
    function Get_Items: IStrings; safecall;
    procedure Set_Items(const Value: IStrings); safecall;
    property Items: IStrings read Get_Items write Set_Items;
  end;
```

```
{ DispInterface declaration for Dual Interface IIStr }

  IIStrDisp = dispinterface
    ['{E6F9F3B7-FD3C-11D0-908F-00C04FC291A4}']
    property Items: IStrings dispid 1;
  end;

{ IStrObject }

  CoIStr = class
    class function Create: IIStr;
    class function CreateRemote(const MachineName: string): IIStr;
  end;



implementation

uses ComObj;

class function CoIStr.Create: IIStr;
begin
  Result := CreateComObject(Class_IStr) as IIStr;
end;

class function CoIStr.CreateRemote(const MachineName: string): IIStr;
begin
  Result := CreateRemoteComObject(MachineName, Class_IStr) as IIStr;
end;


end.
{---------------------------------------------------------------------}
unit Unit1;

interface

uses
  ComObj, Project1_TLB, StdVCL, Classes, AxCtrls, Unit2;

type
  TIStr = class(TAutoObject, IIStr)
  private
    FItems: TStrings;
  public
    destructor Destroy; override;
    procedure Initialize; override;
    function Get_Items: IStrings; safecall;
    procedure Set_Items(const Value: IStrings); safecall;
  end;

implementation

uses ComServ, Dialogs;
```

```
procedure TIStr.Initialize;
begin
  Inherited Initialize;
  FItems := TStringList.Create;
end;

destructor TIStr.Destroy;
begin
  FItems.Free;
end;

function TIStr.Get_Items: IStrings;
begin
  Form1.Memo1.Lines.Assign(FItems);
  GetOleStrings(FItems, Result);
  Form1.StatusBar1.SimpleText := 'TIStr.Get_Items: IStrings';
end;

procedure TIStr.Set_Items(const Value: IStrings);
begin
  SetOleStrings(FItems, Value);
  Form1.Memo1.Lines.Assign(FItems);
  Form1.StatusBar1.SimpleText := 'TIStr.Set_Items(const Value: IStrings)';
end;

initialization
  TAutoObjectFactory.Create(ComServer, TIStr, Class_IStr, ciMultiInstance);
end.
{-----------------------------------------------------------------------}
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, ToolWin, StdVCL, AxCtrls;

type
  TForm1 = class(TForm)
    StatusBar1: TStatusBar;
    Memo1: TMemo;
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
{-----------------------------------------------------------------------}
unit AutCli;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, ToolWin, StdVCL, AxCtrls, Project1_TLB;

type
  TForm1 = class(TForm)
    ToolBar1: TToolBar;
    tbLinestoIStrings: TToolButton;
    tbIStringsToLines: TToolButton;
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure tbLinestoIStringsClick(Sender: TObject);
    procedure tbIStringsToLinesClick(Sender: TObject);
  public
    MyIStr: IIStr;
  end;

var
  Form1: TForm1;

implementation

uses ComObj;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyIStr := CoIStr.Create;
end;

procedure TForm1.tbLinestoIStringsClick(Sender: TObject);
var
  TempIStrs: IStrings;
begin
  GetOleStrings(Memo1.Lines, TempIStrs);
  MyIStr.Set_Items(TempIStrs);
end;

procedure TForm1.tbIStringsToLinesClick(Sender: TObject);
var
  TempIStrs: IStrings;
begin
  TempIStrs := MyIStr.Get_Items;
  SetOleStrings(Memo1.Lines, MyIStr.Items);
end;

end.
{-----------------------------------------------------------------}
```

So why was Unit1, the interface implementation created? An Ole interface, such as IStrings, can be considered a contract that properties and functions will be defined in a given format and that the functions will be implemented as defined (see Developer's Guide, Chapter 36, "An Overview of COM" for more information. The fact that you define an interface does not provide implementation. For

example, to make a defined IStrings interface useable you need to provide storage for the strings and mechanism to add and retrieve strings. Any addtional functionality, defined in the interface, such as a Clear or IndexOf method, must also be implemented.

You may notice in Unit1 that we do not override the Create constructor as the point to create an instance of the FItems TStrings property. Instead we override the Inialize method. The reason for this can be found in the hierarchy of components starting with our TIStr object.

```
TComObject
  TTypedComObject
    TAutoObject
      TISTR
```

In this series of objects, only TComObject has a Create method, and that method IS NOT VIRTUAL. The Create method calls the TComObject.CreateFromFactory and it in turn calls the Initilize method which IS virtual. You can see more detail of the mechanics of this hierarchy in the \source\rtl\sys\comobj.pas unit.

Getting Version Information From Your Program

NUMBER : 3241
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : August 11, 1997

TITLE : Getting Version Information From Your Program

This sample project demonstrates how to get at the version info
in executables.

1. Launch Delphi 3.

2. In a new project, drop a TMemo and a TButton on the form (make sure they're called Memo1 and Button1). Double click on Button1. Now you can go ahead and directly replace the code for the Button1Click procedure (see below).

The Button1Click procedure

```
procedure TForm1.Button1Click(Sender: TObject);
const
  InfoNum = 10;
  InfoStr : array [1..InfoNum] of String =
    ('CompanyName', 'FileDescription', 'FileVersion', 'InternalName',
     'LegalCopyright', 'LegalTradeMarks', 'OriginalFilename',
     'ProductName', 'ProductVersion', 'Comments');
var
  S            : String;
  n, Len, i : Integer;
  Buf          : PChar;
  Value        : PChar;
begin
  S := Application.ExeName;
  n := GetFileVersionInfoSize(PChar(S),n);
  if n > 0 then begin
    Buf := AllocMem(n);
    Memo1.Lines.Add('FileVersionInfoSize='+IntToStr(n));
    GetFileVersionInfo(PChar(S),0,n,Buf);
    for i:=1 to InfoNum do
      if VerQueryValue(Buf,PChar('StringFileInfo\040904E4\'+
                                 InfoStr[i]),Pointer(Value),Len) then
        Memo1.Lines.Add(InfoStr[i]+'='+Value);
    FreeMem(Buf,n);
  end else
    Memo1.Lines.Add('No FileVersionInfo found');
end;
```

you received with the Borland product to which this information pertains.

Delphi Titles from Leading Book Publishers

NUMBER : 8520
PRODUCT : Delphi
VERSION : All
OS : Windows/Win32
DATE : August 21, 1996

TITLE : Delphi Titles from Leading Book Publishers

BORLAND'S DELPHI IS
SUPPORTED BY THESE TITLES
FROM LEADING BOOK PUBLISHERS
April 1996

Delphi Version 2.0

The following is a list of book titles for Delphi 2.0 that are currently available or will be available in the future. For more information, contact the publisher.

* Delphi 2 Developer's Guide; X. Pacheco/S. Teixeira; Borland Press/Sams Publishing; ISBN: 0-672-30914-9
  Phone: 800-428-5331; Fax: 800-448-3804
* Borland's Official No-Nonsense Guide to Delphi 2; M. Manning; Borland Press/Sams Publishing; ISBN: 0-672-30871-1;
  Phone: 800-428-5331; Fax: 800-448-3804
* Teach Yourself Delphi 2 in 21 Days; D. Osier; Borland Press/Sams Publishing; ISBN: 0-672-30863-0;
  Phone: 800-428-5331; Fax: 800-448-3804
* Database Developer's Guide with Delphi 2; K. Henderson; Borland Press/Sams Publishing; ISBN: 0-672-30862-2;
  Phone: 800-428-5331; Fax: 800-448-3804
* Delphi 2 Unleashed, Second Edition; C. Calvert; Borland Press/Sams Publishing; ISBN: 0-672-30858-4
  Phone: 800-428-5331; Fax: 800-448-3804

The New Delphi 2 Programming Explorer; J. Duntemann/J. Mischel/D. Taylor; Coriolis Group; ISBN: 1-883577-72-1;
Phone: 800-410-0192; Fax: 602-483-0193

Delphi 2 Nuts and Bolts, 2nd Edition; G. Cornell; Osborne/McGraw-Hill; ISBN: 0-07-88203-3
Phone: 800-227-0900; Fax: 510-549-6603

Special Edition Using Delphi, 2nd Edition; J. Matcho; Que;
ISBN: 0-7897-0591-5; Phone: 800-428-5331; Fax: 800-882-8583

Delphi 2 By Example, 2nd Edition; B. Watson; Que;
ISBN: 0-7897-0592-3; Phone: 800-428-5331; Fax: 800-882-8583

Building Delphi 2.0 Database Applications; P. Kimmel; Que; ISBN: 0-7897-0492-7; Phone: 800-428-5331; Fax: 800-882-8583

Mastering Delphi for Windows 95; M. Cantu; Sybex;
ISBN: 0-7821-1860-7; Phone: 880-227-2346; Fax: 510-523-2373

Revolutionary Guide to Delphi 2 Programming; B. Long/B. Swart/et al; Wrox Press Ltd.; ISBN: 1-874416-67-2;

Phone: 312-465-3559; Fax: 312-465-4063

* Part of the Borland Press Series

Delphi Version 1.0
---

* Delphi Developer's Guide; X. Pacheco/S. Teixeira; Borland Press/Sams Publishing; ISBN: 0-672-30704-9
* Teach Yourself Database Programming with Delphi in 21 Days; Nathan and Uri Gurewich; Borland Press/Sams Publishing; ISBN: 0-672-30851-7

Delphi Programming Unleashed; C. Calvert; Sams Publishing; ISBN: 0-672-30499-6
The Delphi Programmer Explorer; J. Duntemann/J. Mischel/D. Taylor; Coriolis Group; ISBN: 1-883577-25-X
Delphi Programming for Dummies; N. Rubenking; IDG Books; ISBN: 1-56884-200-7
Foundations of Delphi Programming; T. Swan; IDG Books; ISBN: 1-56884-347-X
Delphi: A Developer's Guide; V. Kellen/B. Todd; MIS Press; ISBN: 1-55851-455-4
Teach Yourself Delphi; D. Hall; MIS Press; ISBN: 1-55828-390-0
Delphi Nuts and Bolts; G. Cornell/T. Strain; Osborne/McGraw-Hill; ISBN: 0-07-882-136-3
Using Delphi, Spec. Ed; J. Matcho/M. Andrew/et al; Que; ISBN: 1-56529-823-3
Mastering Delphi; M. Cantu; Sybex; ISBN: 0-7821-1739-2
Borland Delphi How-To; G. Frerking/W. Niddery/N. Wallace; Waite Group Press; ISBN: 1-57169-019-0
Developing Windows Applications Using Delphi; P. Penrod; John Wiley and Sons; ISBN: 0-471-11017-5
Delphi By Example; B. Watson; Que; ISBN: 1-56529-757-1
Teach Yourself Delphi in 21 Days; A. Wozniewicz; Sams Publishing; ISBN: 0-672-30470-8
Instant Delphi; D. Jewell; Wrox Press Ltd.; ISBN: 1-874416-57-5
Delphi for Windows Power Toolkit; H. Davis; Ventanna; ISBN: 1-56604-292-S

* Part of the Borland Press Series

Adding ODBC Drivers in Delphi 3.0

NUMBER : 3217
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : July 7, 1997

TITLE : Adding ODBC Drivers in Delphi 3.0

Adding ODBC Drivers in Delphi 3.0

The minimum requirements necessary to install an ODBC driver in Delphi 3.0 are as follows:

Microsoft ODBC Manger
Windows 95 or NT
Delphi Developer or Delphi Client/Server edition
Vendor-provided ODBC driver (already installed on your system)

When using Delphi 3.0 there are now two common methods to add ODBC drivers to the BDE. The first step to using any of the methods is to first obtain and install the vendor-supplied ODBC driver onto your system. Once this first essential step is completed the next steps are fairly simple.
In the left panel on the screen is a list of drivers and data sources that have been previously configured for use with BDE applications.

Method A:

1. First start the BDE Administrator from the Windows Start Menu (it will be in the Delphi 3.0 folder.)

2. Now select Object|ODBC administrator from the main menu. (this will bring up a list of currently installed drivers.)

3. Choose Add and then select the ODBC driver you would like to create a data source for, then click on OK.

4. Next fill in the appropriate information for your driver. (A minimal configuration will require the Data Source Name field. You will also need to fill in at least one other field that is a location specifier for the data. This could be a path in the case of Paradox or dBase tables or the Server field in the case of configuring an Interbase ODBC driver. Some non-exhaustive examples include; if you are using Interbase you would select a path to a .GDB file, if using Paradox or dBASE files you would specify the data directory containing your tables or if you were using Oracle you would specify the entry as it appears in your TNSNAMES.ORA file. Once this is done you have created a virtual driver and will be able to access your database files through the datasource your created.)

Method B:

1. First start the BDE Administrator from the Windows Start Menu
(it will be in the Delphi 3.0 folder.)
2. Click on the database tab, then right click inside of the left panel.
3. Select New from the popup menu and then select the type of ODBC driver
you would like to add and click OK.
4. Again right click in the database panel and select Apply from the
popup menu.
5. Now you must select a valid ODBC DSN (Data Source Name) from the
definition panel on the right and select apply.

Both of these methods result in the ability to hook a TDataset in Delphi
to live data.

You may have noticed some new options from the Object|Options
menu, these options allow you to select different configuration modes
to view. It's advisable to have all check boxes in the Select
Configuration Modes to View panel selected. When all the check boxes
are selected you will be provided with an extensive list of all drivers
and aliases available for your use. Without checking the 'virtual'
setting you will not be able to 'see' the drivers that you have added
through the MS ODBC manager but not explicitly through the BDE (as per
method 2).

Developer Support & Customer Support Phone Numbers

NUMBER : 9605
PRODUCT : Borland
VERSION : All
OS : All
DATE : June 5, 1997

TITLE : Developer Support & Customer Support Phone Numbers

The following is a listing of Borland Developer Support and Customer Support phone numbers. For detailed information on the Technical Support programs, please refer to Technical Information Sheet #9800, "Borland Assist". For information on Paradox support, refer to the section below "Obtaining Paradox Support from Corel". For information on ReportSmith support, refer to the section below "Obtaining ReportSmith Support from Strategic Reporting Systems, Inc.".

Developer Support Telephone Numbers

For information about Developer Support telephone numbers, please contact Borland Assist at 1-800-523-7070 (7 a.m. to 4 p.m. Pacific Time, Monday through Friday).

Online Support and Automated Support Phone Numbers

Borland provides the following online and automated support services:

- Borland World Wide Web Site www.borland.com
- Borland Internet FTP site ftp.borland.com
- TechFax Automated Fax Retrieval System 1-800-822-4269
- Borland Download BBS (up to 9600 baud 8N1) 408-431-5096

For more information about these services, see Technical Information Sheet #9800, "Borland Assist", or call 1-800-523-7070 (7 a.m. to 4 p.m. Pacific Time, Monday through Friday).

Borland Assist and Customer Support Phone Numbers

For detailed information on Borland Assist, refer to Technical Information Sheet #9800, or call 1-800-523-7070 (7 a.m. to 4 p.m. Pacific Time, Monday through Friday).

To order the Borland products

Call End-User Sales 1-800-932-9994
(24 hours, 7 days a week)

For inquiries about net terms purchase orders

Call End-User Customer Service 408-461-9190

For all other customer service inquiries

Call End-User Customer Service 510-354-3828

Obtaining Paradox Support from Corel

Corel is licensing Paradox from Borland and now assumes responsibility for Paradox development, support, sales, and marketing.

For general questions about technical support for Paradox, contact Corel at 1-800-772-6735 or visit Corel's Web site at: www.corel.com

---

Obtaining ReportSmith Support from Strategic Reporting Systems

Strategic Reporting Systems, Inc. is licensing ReportSmith from Borland and now assumes responsibility for worldwide service, maintenance, support, marketing, an sales of the ReportSmith family of products. For general questions about technical support for ReportSmith, contact Strategic Reporting Systems at 1-508-531-0905.

Other References

The following Technical Information Sheets provide additional information on Borland Technical Support services. You can order get these documents from TechFax, Borland Download BBS, the Borland web site (www.borland.com), and the Borland FTP site (ftp.borland.com).

TI	Title
9604	An Overview Of Borland Online Information Services
9652	Borland International's TechFax System
9656	Technical Support Via The Internet
9800	Borland Assist

Borland International's TechFax System

NUMBER : 9652
PRODUCT : Borland
VERSION : All
OS : All
DATE : April 23, 1996

TITLE : Borland International's TechFax System

Borland International maintains a library of documents that pertain to specific technical topics (i.e., common questions and answers, applications, and programs) and current Borland news (e.g., product fact sheets, press releases, availability listings, etc.). With TechFax, you can order these documents and have them sent directly to your fax machine.

TechFax is a 24-hour automated service that sends free technical and company information to your fax machine. To access TechFax, call:

800-822-4269

from any touch-tone telephone. You will be greeted and directed to the documents you desire. Simply follow the voice routing menu system and respond to the prompts. Once you become proficient with TechFax, you can even type ahead of the prompts without waiting for them to play in their entirety. You can request up to three documents per call.

The Main Menu:

The main TechFax menu consists of three choices:

Press 1 Information on the TechFax system.

Pressing 1 provides you with a condensed verbal version of this document.

Press 2 To order a document.

If you already know the document number associated to the document you wish to order, press 2 to order the document.

Press 3 To order a TechFax catalog.

If you have not previously used this service, you will first want to order the TechFax information catalog. This catalog provides a global list of product/interest catalogs. These smaller catalogs list specific documents available for particular Borland products and various company news documents.

Each catalog consists of a list of document numbers and brief descriptions of the information presented in each document. A date is also provided to show when a document has been added or updated in the catalog. The documents appear in each catalog sorted by this date so newer documents always appear at the beginning of the catalog.

Press 4 To cancel an order.

Note: You can press **0 (star, star, zero) at any time to return to the main menu.

Ordering a Document:

To order documents, call TechFax at 800-822-4269. Pressing 2 from the main voice menu will bring you to the document ordering prompts. You will be asked to enter up to 3 document numbers. The document numbers must to be separated by the asterisk (*) symbol and finished with a pound sign (#).

For example, if you wished to order documents 500, 793 and 542, you would respond:

500*793*542#

If you are not sure of the document number or don't know what documents are available, order one of the catalog documents listed below.

You can use your touch-tone telephone to request up to three documents per call. TechFax will attempt to successfully send each requested document five time before aborting the request.

Ordering a Catalog:

There are two ways you can order a TechFax catalog. First, from the list of available product/interest catalogs listed below, locate the document number associated with that catalog. Then from the main TechFax menu, press 2 to order the catalog. Second, if you are not sure of the document number of the catalog you wish to order, press 3 from the main TechFax menu. This will present you with a verbal prompts for the catalogs you wish to order.

Available Product/Interest Catalogs:

Document Number	Catalog Name
1	Catalog of available product catalogs
2	Borland News
3	Borland and Turbo C, C++, Brief, and BVT
4	Paradox for DOS
5	Client/Server (InterBase and ReportSmith)
6	dBASE for Windows, Visual dBASE

8 dBASE for DOS, UNIX and VMS
9 Consultants, training centers, 3rd party vendors, and user groups
10 Paradox for Windows
11 Paradox Engine (C and Pascal)
12 SQL Link
14 Delphi
15 ObjectVision
16 Turbo Pascal
20 Turbo Assembler, Debugger, and Profiler
24 Borland Product Information

Personal Identification:

When ordering documents from TechFax, an automated computer service processes your request. If you will be receiving the documents through a central fax machine that is used by others, you will want to identify yourself as the recipient of the documents.

After you have placed your request for documents or catalogs, you will be prompted to enter a personal identification (such as your name). TechFax will place this personal identification on the cover sheet of your order.

Use the keypad on your touch-tone phone as if it were a typewriter that supports three letters on each key. By pressing a key once, the first letter on the key is entered. Pressing the key twice quickly in succession enters the second letter on the key. Pressing the key three times in rapid succession enters the third letter on the key. For the letters Q and Z, use the 1 (one) key, pressing it once for Q and twice for Z. To add a space, press zero. Once you have entered your name as you want it to appear on the fax cover sheet, press the pound key to conclude your entry. As you press keys on the keypad, TechFax will echo the letter it recorded back to you.

If you decide not to use a personal identification for the fax cover sheet, simply press the pound sign when prompted.

TechFax Limitations:

Due to the automated process used to transmit documents, TechFax cannot accept international phone numbers. Technical Information Sheets are also available on the Borland Download BBS,

CompuServe, and on www.borland.com. See the information that follows for information on these services.

Download BBS:

If you are using a FAX modem to receive documents from TechFax and are experiencing difficulties, you may wish to consider using Borland's Technical Support Download BBS. This multi-line download service contains all documents available through

TechFax, as well as other miscellaneous programs and applications. To access the BBS, call (408) 431-5096 from your modem. The settings of your modem are automatically detected so no special setup is required. The service supports modem baud rates up to 9600 bps. For additional information on the Technical Support Download BBS, request document number 9677.

CompuServe:

Technical Information Sheets are also available in the Borland Forums on CompuServe. TIs are located in the following sections of the following forums:

Forum	Section(s)
BCPP	2
BDEVTOOLS	4,7
DELPHI	2
DBASEWIN	10
DBASE	10
PDOXWIN	1
PDOXDOS	1

For more information about Borland Online services, refer to Technical Information Sheet #9604.

www.borland.com:

Technical Information Sheets are also available on the Borland FTP site, which is accessible from www.borland.com. To access Technical Information Sheets for a particular product, click the Tech Info option on the Borland Home Page, then select the product. You can also access the Borland FTP site directly at ftp://ftp.borland.com/pub/techinfo/techdocs/ although, it is easier to locate Technical Information Sheets via the Tech Info pages on www.borland.com.

Borland Fast Fax:

Borland Fast Fax is an automated FAX retrieval system that you can use to request and receive information about Borland and its products. You can order up to five documents at one time. Fast Fax provides the following information:

- Borland product information
- Lists of Developers, System Integrators and Training Companies
- Reseller information
- Educational customer and reseller information
- Borland corporate programs
- Borland Connections

Call 1-800-408-0001

Technical Support Via the Internet

NUMBER : 9656
PRODUCT : Borland
VERSION : All
OS : All
DATE : April 25, 1996

TITLE : Technical Support Via the Internet

Users of all Borland products now have the option to retrieve technical resources and information via 'anonymous' File Transfer Protocol (FTP) over the Internet. FTP is the Internet standard for transferring files to and from a remote network host.

This service will connect millions of users in the global user-base of Borland products and the Internet community together. This adds yet another option and method for our users to obtain world-class technical support. Through the Internet and anonymous FTP, users will have the ability to directly access technical information and obtain world-class technical services and support.

ACCESS TO FTP

Most users of the Internet community have the ability to take advantage of ftp.borland.com. The examples supplied in this document assume you have:

1. access to the proper Internet connection
2. availability to FTP services on a UNIX operating system

See *NOTE below.

You can also access ftp.borland.com from www.borland.com. The easiest way to access technical information for Borland products is to select the Tech Info page from www.borland.com for a particular Borland product. To do this select Tech Info from the Borland home page (www.borland.com), then choose the product. On the Tech Info page for each Borland product there is an option for FTP Files and for Technical Information.

Connecting to ftp.borland.com

To connect to ftp.borland.com, you can either type 'ftp' or 'ftp ftp.borland.com' at the UNIX prompt. Either way, you will be connected to the remote FTP server. UNIX system prompts and responses are enclosed in {} braces.

```
{UNIXPrompt%} <----- TYPE ftp ftp.borland.com
{Connected to ftp.borland.com.}
{220 ftp FTP server (SunOS 4.1) ready.}
```

{Name (ftp.borland.com:<username>):} <- TYPE anonymous
{311 Guest login ok, send ident as password.}
{Password:} <--- TYPE your Internet mailing address here
{230 Guest login ok, access restrictions apply.}
{ftp>}

--or--

{UNIXPrompt%} <----- TYPE ftp
{ftp>} <----- TYPE open ftp.borland.com
{Connected to ftp.borland.com.}
{220 ftp FTP server (SunOS 4.1) ready.}
{Name (ftp.borland.com:<username>):} <- TYPE anonymous
{311 Guest login ok, send ident as password.}
{Password:} <--- TYPE your Internet mailing address here
{230 Guest login ok, access restrictions apply.}
{ftp>}

FTP COMMANDS
--- --------
The core commands of the File Transfer Protocol (FTP) are supplied below. Examples using some of these commands will be included later in this document.

Command	Function
ls	List contents of remote directory
dir	List contents of remote directory
cd <directory>	Change remote directory (same as DOS)
lcd <directory>	Change local directory
cdup	Change dir to remote parent dir
pwd	Print working directory of remote host
prompt	Force interactive prompting on multiple commands (default is on)
get <filename>	Receive <filename> to local system
mget <filename1 filename2 ...>	Receive multiple files to local system
open <remote host>	Opens a session with remote host
ascii	Set ASCII transfer type
binary	Set binary transfer type for executable, data, compressed, or object files
status	Show current status
!	Shell to local operating system
quit	Terminate FTP session and exit

THINGS TO KNOW:
****** ** *****

File Naming Conventions

---- ------ -----------
This is a UNIX application. There are two concerns to be aware of:

1. File names are case sensitive.
2. If you are copying files to your local machine, and it uses the DOS file naming convention, it will truncate the filename.

   Example: TI1051111.txt
   TI1051112.txt
   TI1051113.txt --> in DOS are all
   TI105111.txt

HELP

In the FTP application, you can type 'Help' or '?' for a listing of commands. For help on a particular command:

TYPE help <command name> or ? <command name>

INDEX Files
----- -----
Borland offers users files in the /pub directory that corresponds to the files and resources available to you. It is recommended that you download the INDEX and readme text file(s) before retrieving documents. This will save you a great deal of time searching for files and information. The exercise steps you through retrieving these helpful and important files.

Wildcards

You can use the wildcard (*) when specifying (a) filename(s) in commands.

Command Status
------- ------
When the user issues a command at the FTP prompt, the application will respond with its current status. For example:

command successful
?Invalid command -or- ?Ambiguous command
Transfer complete.
ASCII Transfer complete.
Binary Transfer complete.

FILE TYPE: Binary vs. ASCII
---- ----- ------ --- -----
Remember to set the file type before sending and retrieving files. If you do not specify the correct file type, you will not

be able to correctly use the file.

Use Binary file transmission for files with extension:
.zip, .exe, .com
Use ASCII file transmission for files with extension:
.dat, .txt, .pas, .c

NOTE: If you are not sure which file type to use, set file type
to Binary.

.ZIP Files

Many files at ftp.borland.com are saved in compressed .ZIP format. They must be uncompressed with PKUNZIP before they can be used or viewed properly. The files have been compressed with PKZIP version 2.04g. The /pub/libs/misc/gen directory contains the self-extracting .EXE version of this utility program.

Abbreviations:

BGI - Borland Graphics Interface Files
BI - Basic Information. General files. Additional product samples.
TI - Technical Information sheets. TIs have been created to further assist our users accomplish their tasks at hand. Tips and Tricks! TIs offer an additional resource that compliments and builds upon the information presented in our manuals and product HELP.
TL - Template Language for dBASE
TFMS - Text Font Metrics Files
TV - Turbo Vision
QnA - Questions and Answers

HELPFUL EXAMPLE:
******* ********

Open an FTP session at ftp.borland.com and get the INDEX document in the directory /pub. Then mget filxdir.txt and readme.txt.

* To change directory, type:

{ftp>} cd /pub
{250 CWD command successful}

* To get a file, type:

{ftp>} get INDEX
{200 PORT command successful.}
{150 ASCII data connection for INDEX (272775 bytes).}
{226 ASCII Transfer complete.}
{272775 bytes received in 16.69 seconds (16.25 Kbytes/s)}
{ftp>}

* To get a file and save it by another name in the local directory. The syntax is:

get <filename> <new file name>

For example:

{ftp>} get INDEX MYINDEX.TXT

* To get multiple files, type:

{ftp>} mget filxdir.txt readme.txt
{mget filxdir.txt?} y <- y for yes / n for no
{200 PORT command successful.}
{150 ASCII data connection for filxdir.txt(57727 bytes).}
{226 ASCII Transfer complete.}
{58954 bytes received in 3.36 seconds (17.13 Kbytes/s)}
{mget readme.txt?} y
{etc.}

* To quit, type: {ftp>} quit

****

*NOTE: Although not included nor discussed in this document, there are many programs and applications available that will allow PC users, who have the proper Internet connection, to directly access FTP sites.

PROBLEMS

Problems accessing information on ftp.borland.com should be reported to roger@borland.com. Do NOT e-mail messages that do not concern this system. Please include "SYSTEM PROBLEM" in the subject of your message.

Borland is currently supporting 67 ports on this site 24 hours, 7 days a week. Depending on the popularity and usage of this service, this number of connections may increase. Please try again if the connections are busy.

Borland's Technical Suport Download Bulletin Board

NUMBER : 9677
PRODUCT : Borland
VERSION : All
OS : All
DATE : April 25, 1996

TITLE : Borland's Technical Suport Download Bulletin Board

Borland International maintains a library of documents, applications and programs that pertain to specific technical topics and current Borland news. Through the Technical Support Download BBS, you can download these documents and files to your computer.

The Technical Support Download BBS is a multi-line 24 hour download service that allows you to retrieve free technical information. To access the BBS, from your modem call:

(408) 431-5096

The settings of your modem are automatically detected so no special setup is required. However, if you encounter any difficulties accessing the service, set your modem to use 8 data bits, no parity, and 1 stop bit (8-N-1). The Download BBS supports modem baud rates up to 9600 bps.

When you access the BBS you will be greeted with a easy to follow menu system. Select the area of interest from the menu and you will see a list of available files and a brief description for each file. You can download any of the listed files.

Technical information available through the Download BBS include documents discussing technical product information, tips & traps, common questions & answers and sample programs & applications. Current Borland news include press releases, product fact sheets, announcements, product availability listings and product comparisons.

Borland Assist

NUMBER : 9800
PRODUCT : Borland
VERSION : All
OS : All
DATE : June 5, 1997

TITLE : Borland Assist

Borland is committed to providing you with the best possible services and assistance. In keeping with this commitment, we are proud to introduce the NEW Borland Assist developer support program. This new program is packed with expert support, exceptional value, and extra-ordinary flexibility.

Whether you're an occasional user or a power user, Borland is ready to assist you. To order any or Borland's Assist programs, please call Borland Assist at 800 523-7070.

The new developer support structure includes several new programs differentiated by levels of support, rather than by product lines. This will provide a unified process for speeding response times and reducing the number of support contracts needing to be managed by customers.

Beginning May 1, 1997, Borland will begin taking orders for its new support contracts. Borland will, at the same time, continue to honor all existing contracts. The new support programs apply to customers in the U.S. and Canada, and are scheduled to be in operation by July 1, 1997. To contact a Borland Sales representative for more information on these support programs, please call (408) 431-1064.

Support hours for all Borland products will be 7 am to 4 pm., Pacific Standard Time, starting June 4, 1997.

New Developer Support Services for the U.S. and Canada

Following is an overview of new developer support services. Detailed descriptions of each of the services appears below this overview.

Installation Assist -- Offers Borland customer support via telephone on Borland workstation software products. Questions will be answered concerning local installations during the first 90 days after the first call to Borland's service center. This service is scheduled for availability starting May 1, 1997.

Primary Assist -- Customers of Borland workstations software products can call Borland for per-minute telephone support on local installation and product usability. This service is scheduled for availability starting June 30, 1997.

Developer Incident Assist -- Customers receive support via telephone for questions concerning installation, programming, connections to database servers, and usability of Borland products in a workstation, network, or client/server environment on a per incident basis. This service is scheduled for availability starting June 15, 1997.

Priority Developer Assist -- Provides customers an annual developer support service with priority hotline assistance during service hours on all supported Borland workstation products. The service covers questions concerning the installation, programming, connections to database servers, and usability of Borland products in a workstation, network, or client/server environment for a predefined number of 15 incidents or 12 months. This service is scheduled for availability starting May 1, 1997.

Extended Developer Assist -- Offers customers priority hotline services during service hours on all supported Borland workstation products, including extended products such as Delphi/400. Questions concerning the installation, programming, connections to database servers, and usability for Borland products in a workstation, network, or client/server environment are covered for a predefined number of 15 incidents or 12 months. This service is scheduled for availability starting May 1, 1997.

Description of activities
INSTALLATION ASSIST

Service Description
Installation Assist offers Borland customers technical support via telephone on registered Borland products listed on the Supported Products List. Questions will be answered concerning the local installation of Borland products on workstations during the first 90 days after the first call on a product to the service center.

Service Features
* Support via telephone
* Access to the Developer Support area on the WWW, Borland Online.
* Access to Technical support personnel.
* Charge-free installation assistance for registered Borland customers for 90 days, standard telephone toll charges apply.
* For customers of all Borland products covered by Installation Assist.
* Questions related to the Borland product on local installation and opening sample tables are supported under this service
* Questions on usability issues, programming, connecting to database servers, database servers, pre-releases, trial versions, Borland OED products and products on the Extended Product List are not supported under this service.
* Customer will be given a support reference number to be used in consecutive calls on the same issue,
* Installation Assist is provided for owners of products on the

Supported Product List. This list is available on the WWW, and is updated periodically. The list is also available from Borland Customer Service upon request.

Customer Responsibilities

* Support will only be given for products that have been registered by the customer.
* Customers should first read the on-line text and readme files; these files may include information that make it unnecessary to call for support.
* The customer must provide a clear, detailed description of the problem or the question.
* Installation Assist be provided only for Borland products used in accordance with the defined hardware and operating system requirements.
* Customer must be at the workstation where the product is being installed at time of call.

Duration

* The no charge Installation Assist is provided for up to 90 days after the first call to the service center on a supported Borland product.

Service Hours

* Service hours are from 7 am to 4 pm Pacific time from Monday through Friday on business days.

Terms and Conditions

* All services are provided subject to the current Borland Developer Support Program's Terms and Conditions (see the section at the end of this document).

Description of activities
PRIMARY ASSIST

Service Description
Primary Assist offers Borland customers technical support via telephone on registered Borland products listed on the Supported Products List. A single issue will be answered concerning the local installation or usability of Borland products on workstations.

Service Features

* Support via telephone.
* Access to the developer support area on the WWW, Borland Online.
* Access to technical support personnel.
* Questions related to the Borland product on installation, user interface, programming syntax, opening local sample tables and explanation of error messages are supported under this service.
* Questions on customer's application code, connecting to database server, database servers, pre-releases, trial versions, Borland OED products and products on the Extended

Product List are not supported under this service.
* Customer will be given a support reference number to be used in consecutive calls on the same issue
* Primary Assist is provided for owners of products on the Supported Product List. This list is available on the WWW, and is updated periodically. The list is also available from Borland Customer Service upon request.
* Primary Assist is delivered for a single incident or question.
  - an incident is defined as a single support issue with a Borland product and the reasonable effort needed to resolve it. A single support issue is a problem that cannot be broken down into subordinate parts. Before Borland responds to an incident, the customer and Borland's engineer must agree on what the problem is and the parameters for providing a resolution
  - it is possible for one incident to span multiple telephone calls, it is also possible for one telephone call to include multiple incidents
  - when, after 3 attempts on separate business days, the customer cannot be reached regarding an open incident, the incident is considered closed, unless otherwise agreed by Borland and customer.

Customer Responsibilities
* Support will only be given for products that have been registered by the customer.
* The customer should provide a clear, detailed description of the problem or the question.
* Support will be provided only for Borland products used in accordance with the defined hardware and operating system requirements.
* Customer must be at the workstation where the product is installed at time of call.

Service Hours
* Service hours are from 7 am to 4 pm Pacific time from Monday through Friday on business days.

Support Fee
* The price of Primary Assist is $2.95 per minute, with a mimimum charge of $15, or a fixed price of $95. The per-minute cost is calculated as the sum of phone minutes, off-line research and possible call back. The customer and Borland's engineer must agree on what pricing model will be used before Borland will accept the call.

Terms and Conditions
* All services provided are subject to the current Borland Developer Support Program's Terms and Conditions (see the section at the end of this document).

Description of activities
DEVELOPER INCIDENT ASSIST

Service Description

Developer Incident Assist offers Borland customers developer support on registered Borland products listed on the Supported Product List. A single issue will be answered concerning the installation, connectivity or development of Borland products.

Service Features

* Support via telephone.
* Access to the Developer Support area on the WWW, Borland Online.
* Access to a Borland Developer Support engineer.
* Questions related to the Borland product on installation, programming and debugging, connecting to local and remote databases, and explanation of error messages are supported under this service.
* Questions on customer's application code, pre-releases, trial versions, Borland OED products, products on the Extended Product List and database servers are not supported under this service.
* Customer will be given a support reference number to be used in consecutive calls on the same issue.
* Developer Incident Assist is provided for owners of products on the Supported Product List. This list is available on the WWW, and is updated periodically. The list is also available from Borland Customer Service upon request.
* Developer Incident Assist is delivered for a single incident or question.
  - an incident is defined as a single support issue with a Borland product and the reasonable effort needed to resolve it. A single support issue is a problem that cannot be broken down into subordinate parts. Before Borland responds to an incident, the customer and Borland's engineer must agree on what the problem is and the parameters for providing a resolution
  - it is possible for one incident to span multiple telephone calls, it is also possible for one telephone call to include multiple incidents
  - when, after 3 attempts on separate business days, the customer cannot be reached regarding an open incident, the incident is considered closed, unless otherwise agreed by Borland and customer
* Borland will endeavour to provide resolutions to questions within a reasonable time, but at least within 2 business days after the question has been received. If the nature of the question prevents Borland from providing a resolution within that period, Borland will contact customer to inform customer when a resolution can be expected.

Customer Responsibilities

* Support will only be given for products that have been registered by the customer.
* The customer should provide a clear, detailed description of the problem or the question.
* Support will be provided only for Borland products used in accordance with the defined hardware and operating system requirements.

* Customer should be at the workstation where the product is installed at time of call.

Service Hours
* Service hours are between 7 am to 4 pm Pacific time from Monday through Friday on business days.

Support Fee
* The purchase price for the service is $185.

Terms and Conditions
* All services provided are subject to the current Borland Developer Support Program's Terms and Conditions (see the section at the end of this document).

Description of activities
PRIORITY DEVELOPER ASSIST

Service Description
Priority Developer Assist offers Borland customers an annual contract to obtain developer support on registered Borland products listed on the Supported Product List. Questions will be answered concerning the installation, connectivity and development of Borland products.

Service Features
* Support via telephone with priority access over Basic Assist and Developer Incident Assist.
* No named customer contact necessary.
* Access to the Developer Support area on the WWW, Borland Online.
* Access to a Borland Developer Support engineer.
* Questions related to the Borland product on installation, programming and debugging, connecting to local and remote databases, and explanation of error messages are supported under this service.
* Questions on customer's application code, pre-releases, trial versions, Borland OED products, products on the Extended Product List and database servers are not supported under this service.
* Priority Developer Assist is provided for owners of products on the Supported Product List. This list is available on the WWW, and is updated periodically. The list is also available from Borland Customer Service upon request.
* Priority Developer Assist is provided until the predefined number of 15 incidents have been used or 12 months have elapsed, whatever comes first.
  - an incident is defined as a single support issue with a Borland product and the reasonable effort needed to resolve it. A single support issue is a problem that cannot be broken down into subordinate parts. Before Borland responds to an incident, the customer and Borland's engineer must agree on what the problem is and the parameters for providing a resolution.

- it is possible for one incident to span multiple telephone calls, it is also possible for one telephone call to include multiple incidents
- when, after 3 attempts on separate business days, the customer cannot be reached regarding an open incident, the incident is considered closed, unless otherwise agreed by Borland and customer.

* Borland will endeavour to provide resolutions to questions within a reasonable time, but at least within 2 business days after the question has been received. If the nature of the question prevents Borland from providing a resolution within that period, Borland will contact customer to inform customer when a resolution can be expected.

Customer Responsibilities

* Support will only be given for products that have been registered by the customer.
* The customer should provide a clear, detailed description of the problem or the question.
* Support will be provided only for Borland products used in accordance with the defined hardware and operating system requirements.
* Contracts and customer identification numbers are non-transferrable between different companies or individuals.
* Customer should be at the workstation where the product is installed at time of call.

Contract Duration

* The use of Priority Developer Assist for Borland products is limited to twelve months from the purchase date of the service or until the maximum number of incidents has been exhausted, whatever comes first.
* When twelve months have elapsed, or the number of incidents has been exhausted, the customer will be offered a new annual contract. The twelve month period starts at the first day of the month following the purchase of the Priority Developer Assist annual contract.
* Non-support services, such as the capability to log on-line bug reports and to access product patches and releases will remain in effect even when no incidents are available.

Service Hours

* Service hours are between 7 am to 4 pm Pacific time from Monday through Friday on business days.

Support Fee

* The purchase price for the service is $2495.

Terms and Conditions

* All services provided are subject to the current Borland Developer Support Program's Terms and Conditions (see the section at the end of this document).

Description of activities

EXTENDED DEVELOPER ASSIST

Service Description
Extended Developer Assist offers Borland customers an annual contract to obtain developer support on registered Borland products listed on the Extended Product List. Questions will be answered concerning the installation, connectivity and development of Borland products.

Service Features
* Support via telephone with priority access over Basic Assist and Developer Incident Assist support.
* No named customer contact necessary.
* Access to the Developer Support area on the WWW, Borland Online.
* Access to a Borland Developer Support engineer.
* Questions related to the Borland product on installation, programming and debugging, connecting to local and remote databases, and explanation of error messages are supported under this service.
* Questions on customer's application code, pre-releases, trial versions, and Borland products not on the Extended Supported product List are not supported under this service.
* Extended Developer Assist is provided for owners of products on the Extended Product List. This list is available on the WWW, and is updated periodically. The list is also available from Borland Customer Service upon request.
* Extended Developer Assist is provided until the predefined number of 15 incidents have been used or 12 months have elapsed, whatever comes first.
  - an incident is defined as a single support issue with a Borland product and the reasonable effort needed to resolve it. A single support issue is a problem that cannot be broken down into subordinate parts. Before Borland responds to an incident, the customer and Borland's engineer must agree on what the problem is and the parameters for providing a resolution.
  - it is possible for one incident to span multiple telephone calls, it is also possible for one telephone call to include multiple incidents
  - when, after 3 attempts on separate business days, the customer cannot be reached regarding an open incident, the incident is considered closed, unless otherwise agreed by Borland and customer.
* Borland will endeavour to provide resolutions to questions within a reasonable time, but at least within 2 business days after the question has been received. If the nature of the question prevents Borland from providing a resolution within that period, Borland will contact customer to inform customer when a resolution can be expected.

Customer Responsibilities
* Support will only be given for products that have been registered by the customer.
* The customer should provide a clear, detailed description of the problem or the question.

* In order for the customer to obtain support, the Borland product must be used in accordance with the defined hardware and operating system requirements.
* Contracts and customer identification numbers are non-transferrable between different companies or individuals.
* Customer should be at the workstation where the product is installed. If applicable, the workstation should be connected to the database server via a supported connectivity software at time of call.

Contract Duration

* The use of Extended Developer Assist for Borland products is limited to twelve months from the purchase date of the service or until the maximum number of incidents has been exhausted, whatever comes first.
* When twelve months have elapsed, or the number of incidents has been exhausted, the customer will be offered a new annual contract. The twelve month period starts at the first day of the month following the purchase of the Extended Developer Assist annual contract.
* Non-support services, such as the capability to log on-line bug reports and to access product patches and releases will remain in effect even when no incidents are available.

Service Hours

* Service hours are between 7 am to 4 pm Pacific time from Monday through Friday on business days.

Support Fee

* The purchase price for the service is $3495.

Terms and Conditions

* All services provided are subject to the current Borland Developer Support Program's Terms and Conditions (see the section at the end of this document).

Borland Developer Support Programs
Terms and Conditions

Please read these Terms and Conditions Carefully

1. We will undertake commercially reasonable efforts to provide technical assistance under this agreement, but do not guarantee that all problems will be solved or that any item will be error free.

2. We may, from time to time, discontinue products and versions, stop supporting selected products and versions within a reasonable time after discontinuance, or discontinue any or all support services. We also reserve the right to terminate service to any individual who abuses any support program including, but not limited to sharing special phone numbers and customer identification numbers with others.

3. BORLAND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND REGARDING THE SOFTWARE OR ANY SERVICES WE MAY PROVIDE, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ARISING BY STATUTE, LAW OR TRADE DEALING OR USAGE. ALL MATERIALS AND SERVICES ARE PROVIDED "AS IS."

4. We are not liable for incidental, special or consequential damages for any reason (including loss of time, loss of data or software, loss of profits, or loss of revenue) even if Borland has been specifically advised of the possibility of such damages, and our liability in all events will not exceed the support fees that you have paid under this agreement.

5. We own (including copyrights) all work we do and all information we give to you as part of our support programs. We grant you a non-exclusive license to use that work and information for yourself, or internally within your company, to the extent such use would be permitted in the No-Nonsense License Statement that you received with the Borland product to which this information pertains. We have the right to use and treat as non-confidential any information you may give us during your use of our support program unless you specify in writing the fact that certain material should be treated as being confidential.

6. This is the full and final agreement between you and us, and supersedes any promises, representations or agreements relating to the subject of this agreement. This agreement may only be changed if you and our authorized representative do so in writing. No inconsistent, additional, or pre-printed terms on your purchase order or other business form apply.

Delphi/400: Activating your License Key

NUMBER : 3218
PRODUCT : Delphi
VERSION : All
OS : Windows
DATE : July 7, 1997

TITLE : Delphi/400: Activating your License Key

This document describes the steps to obtain and apply a permanent security key for ClientObjects/400 in the Delphi/400 package. Also refer to the instructions contained in the pamphlet "Additional Information for Delphi/400 Users" that ships with Delphi/400.

Delphi/400 software is installed to both an AS/400 and a PC. When the AS/400 part of the installation is complete, a temporary automatic key of two months(60 days) will be created. After this time functionality will be unavailable, and attempting to access the AS/400 through the supplied components will generate a protection error.

To obtain a permanent key, fill in the registration card that ships with Delphi/400 and follow the mailing directions on the card. Be sure to include your AS/400 model and serial number (these numbers are used to compute a unique key). The model and serial number can be obtained by running the following AS/400 commands:

DSPSYSVAL(QMODEL)
DSPSYSVAL(QSRLNBR)

When you receive your permanent key, activate the key by entering the following commands at an AS/400 terminal.

ADDLIBLE CO400
CALL CO400INS

The screen that appears allows you to enter your permanent key.