



We have a new capability for UNDU... Marc Ferran (100702,1215) from Barcelona, Spain has graciously delved into some of the inner-workings of the Windows Help system for us and has provided two additional files that you will find in with this latest issue. The files are called DNHELP.HLP and DNHELP.CNT

If you are maintaining a directory of all issues of UNDU, simply move these two additional files into that directory. Then, when you want to read the issues, simply double-click on DNHELP. You will get a very nice overview of all issues as well as having the ability to search for articles across all issues. From what I can tell, however, this facility may only work in Windows 95 as it uses the new help engine. Marc, you may want to correct me on this if necessary...

Each issue, I will provide updated copies of these files so they will always stay up to date! **Thanks Marc!!!**

[UNDU Awards](#) ***Please Read!***

[DBExtender Product Announcement](#)

[The Delphi Magazine](#)

[The Beginners Corner](#)

[Core Concepts With Delphi - Part I](#)

[Core Concepts With Delphi - Part II](#)

[Dynamic Form Creation](#)

[Dynamic Delegation](#)

[Tips & Tricks](#)

[The Component Cookbook](#)

[Where To Find UNDU](#)

[Index of Past Issues](#)



Tips & Tricks

[Selecting Objects in the IDE](#) **** WOW!! ****

[Finding Run-Time Errors](#)

[Top Ten Beginners Tips](#)

[Return to Front Page](#)



The Beginners Corner - Talkin the Talk: CLASS or OBJECT

Paul H. Comitz - CIS: 102565,3167

Maybe Borland will bring us 32 bit Delphi soon. I haven't been keeping track of the release date, but I do recall the statement "90 days after the release of Windows 95". It looks like it will be available about the time that you read this. I hope there's support for TAPI (Telephone Application Programming Interface). Then we'll write some programs here to control our telephone equipment. It's an exciting time to be a programmer.

At least part of the excitement is derived from the power of our programming languages. Delphi is designed so that mere mortals can conceive and construct exciting, useful applications. I think at least part of Delphi's power is derived from its marriage to the object paradigm. Try to forget that all the stuff you've heard about Pascal as a structured language for students. A quick peek under the hood reveals Delphi as an object oriented language that rivals all the biggies. Yes, yes, the OOP disciples will respond with a litany about operator and function overloading and multiple inheritance (which Delphi doesn't have). I've never needed to overload an operator and function overloading is only useful if your documentation is complete, up to date, and sitting on your lap. Function overloading IS a super feature if you can communicate what you've done to others! (Are you listening Borland???)

The debate will go on forever. Some say that C and C++ were designed to "trust the programmer", and strongly typed structured languages like Pascal or Ada were designed to "protect the programmer". Advocates of languages like Pascal and Ada say that C allows programmers to put a rope around their neck while they promise not to hang themselves. C programmers reply, with a visible swagger, "but I won't hang myself. Look at all the great stuff I've done already". My two cents: I think Object Pascal includes most of the features that C++ disciples so desperately crave, while providing all the protection of "traditional" Niklaus Wirth Pascal. I think the tradeoff is worth it.

Now that I've got that out of my system lets look at creating our own objects in Delphi. Delphi allows us to create "objects" using the reserved words OBJECT or CLASS. A reasonable person immediately asks "what's the difference?" An unreasonable persons responds " Well, they're the same, but they're really different." Unquestionably, there are no fooling technical differences between OBJECT and CLASS. Another MAJOR consideration to this apparent disconnect, that you won't find in any of the literature, is plain old marketing. Remember that Delphi is a product aimed at programmers and developers. To understand why there are two ways to do "kind of " the same thing, a little background is required.

Somewhere back in the Turbo Pascal days (I think it was version 5.5 which puts us around 1990) Borland began supporting objects in their Pascal compilers. OOP was gaining in popularity, and Pascal was rapidly fading to black. Support of objects was the right thing to do... technically AND economically. Declaration and use of the Turbo Pascal object type reminded some of us of a RECORD that included procedures and functions. A simple object declaration in Turbo Pascal is:

```
type
  Airplane = object
    model :string[10];
    name :string[30];
    procedure initialize ;
```

```
end; (*Airplane*)
```

The structure above defines two variables named *model* and *string*, and a procedure named *initialize*. *This exact code will compile in Delphi*, but its difficult to use, when compared to the new Delphi object model.

Meanwhile back at Borland headquarters, the decision was made to leverage the Turbo Pascal product. Of course, this decision is Delphi. Again recognizing the Delphi is aimed at programmers, Borland decided to go to a full blown 32 bit object model . Additionally Borland decided to use C++ nomenclature to define objects. That nomenclature is the reserved word CLASS. On some emotional level, existence of the reserved word CLASS makes C++ programmers feel better. It's interesting to note that the entry for objects in the index of the "Object Pascal Language Guide" directs one to a chapter called "Class Types". I haven't been able to find ANY references to old style objects in the Guide. That's a major hint folks.

Definition of the structure above using the reserved word CLASS is:

```
type
  TAirplane = class
    model : string[10];
    name : string[30];
    procedure initialize;
end; {TAirplane}
```

For this simple example, it's APPEARS that the same thing is happening. Do a topic search on OBJECT and CLASS. The entry for OBJECT refers to the previous versions of Turbo Pascal as we discussed above. The entry for CLASS states that "the reserved word CLASS is used to declare an object type or class method". In this context an object type refers to the NEW DELPHI OBJECT MODEL that will be the basis for 32 bit Delphi. It IS confusing. There's actually two object models resident in Delphi. The old style 16 bit object model invoked by the reserved word OBJECT and the new Delphi model invoked by the reserved word CLASS. Here's a suggestion: Unless your porting code from a previous Borland Pascal product, simply use the CLASS mechanization all the time. Remember that the reserved word OBJECT is provided for backwards compatibility. If you are porting code, you have to be aware of both mechanizations.

There's an immediate benefit in creating objects using the CLASS reserved word. Recall the code fragment:

```
TAirplane = class
```

We could have also written:

```
TAirplane = class(TObject)
```

This means that the objects we create of type TAirplane inherit all the functionality of the Delphi TObject object. All newly created objects default to TObject as an ancestor unless a different ancestor is explicitly stated i.e.

```
TAirplane = class(TVehicle)
```

assuming a TVehicle object had been defined elsewhere.

The things we've discussed here can tax even the most patient. An example and some experimentation with the debugger and the Object browser go a long way in illustrating these abstract concepts. Here's an exercise to get started. Consider the following:

```

(*****)
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 }
  end;

  {You add this code}
  type
    TAirplane = class(TObject)
      model : string[10];
      name : string[30];
      procedure initialize;
    end; {TAirplane}

var
  Form1: TForm1;
  {You add this code}
  Aircraft1:TAirplane;

implementation

{$R *.DFM}

  {You add this code}
  Procedure TAirplane.Initialize;
  begin

end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {You add this code}
  Aircraft1.model := 'F14';
  Aircraft1.name := 'Tomcat';
end;

end.
(*****)

```

This is just a blank form Delphi project with one button and an event handler for that button. I've marked the code you need to add with the comment *{You add this code}*. Now run the program and press the button.....Sorry about the GPF, I needed to prove it to you. The best way to learn to avoid mistakes is to make the mistake, usually several times. When you pressed the button, the event handler for that button, Button1Click, was invoked. Button1Click accesses the fields of the object Aircraft1. The program runs fine until the Button1 is depressed.

Why the GPF? Don't we create objects and components all the time with the VCL. When the variable

Aircraft1 is declared, we are defining an INSTANCE of the object TAirplane. We haven't created this object yet. Although many objects and components are created automatically for us in Delphi, Objects defined by the programmer MUST BE CREATED MANUALLY.

Recall that we inherit all the functionality of the object TObject. TObject includes two important methods called Create and Destroy. To manually create an instance of the TAirplane Object called Aircraft1, add the following line of code add the following line of code as the FIRST line in the event handler Button1Click:

```
Aircraft1.Initialize;
```

Additionally, add the following to Procedure TAirplane.Initialize:

```
Aircraft1 := TAirplane.Create;
```

We've done two things here. First we call the Initialize method *for the class* TAirplane. Second, the Initialize method of TAirplane calls the inherited (from TObject) method Create. To show the effect of the Create method, single step through the example program with the variable Aircraft1 in the watch window. Make sure you set the watch type to pointer.

If you've never used either of these tools before, make sure you do so now. These tools can be a programmer's best friend. To single step through the program select Trace Into from the Run menu. The program will run. When you press Button1, you'll see the first line of Button1Click highlighted. Repeatedly press F7 to "single step" through the program. To place an item in the watch window select Add Watch from the Run menu and type Aircraft1 into the Expression box.

While single stepping observe Aircraft1 in the Watch window. Initially, the watch window reports 0000:0000. As soon as the Create method in TAirplane.Initialize is executed, an address appears. The instance, Aircraft1, of the object TAirplane, has been created (memory has been allocated for the instance of this object). Additionally, the Create method initializes all the fields of this instance to zero (verify this by placing Aircraft1.model or Aircraft1.name in the watch window). The Create method and methods like it are often referred to as Constructors. You can add to the functionality of the inherited Create method we've illustrated here if you have special creation requirements for your object.

Once you've created an object you are obliged to dispose of it when you are through. Use the online documentation and experiment with the inherited Free and the Destroy methods. In fact, there are 17 methods inherited from TObject. Prove this to yourself by examining TAirplane in the Object Browser.

Here's are the key issues we've discussed:

- 1) There's two separate and distinct object models contained in Delphi.
- 2) Use the CLASS keyword to use the new Delphi Object model
- 3) Objects defined by the programmer must be manually created.

Until next month!

[Return to Front Page](#)



Index of Past Issues

Below is a complete index of all principle articles in past issues of the Unofficial Newsletter of Delphi Users. Provided that you have the prior issues in the same directory as this issue, you can click on any of these hotspots to go directly to that article. To return to the index, you can click on the **Back** button, or you can use the **History** list. Once you jump to one of these issues, you can navigate through the issue as you would normally, but you will need to go to the **History** list to get back to this index. There will be an updated index included in all future issues of UNDU.

[Issue #1 - March 15, 1995](#)

[What You Can Do](#)
[Component Design](#)
[Currency Edit Component](#)
[Sample Application](#)
[The Bug Hunter Report](#)
[About The Editor](#)
[SpeedBar And The ComponentPalette](#)
[Resource Name Case Sensitivity](#)
[Lockups While Linking](#)
[Saving Files In The Image Editor](#)
[File Peek Application](#)

[Issue #2 - April 1, 1995](#)

[Books On The Way](#)
[Making A Splash Screen](#)
[Linking Lockup Revisited](#)
[Problem With The CurrEdit Component](#)
[Return Value of the ExtractFileExt Function](#)
[When Things Go Wrong](#)
[Zoom Panel Component](#)

[Issue #3 - May 1, 1995](#)

[Articles](#)
[Books](#)
[Connecting To Microsoft Access](#)
[Cooking Up Components](#)
[Copying Records in a Table](#)
[CurrEdit Modifications by Bob Osborn](#)
[CurrEdit Modifications by Massimo Ottavini](#)
[CurrEdit Modifications by Thorsten Suhr](#)
[Creating A Floating Palette](#)
[What's Hidden In Delphi's About Box?](#)
[Modifications To CurrEdit](#)

[Periodicals](#)
[Progress Bar Bug](#)
[Publications Available](#)
[Real Type Property Bug](#)
[TIni File Example](#)
[Tips & Tricks](#)
[Unit Ordering Bug](#)
[When Things Go Wrong](#)

[Issue #4 - May 24, 1995](#)

[Cooking Up Components](#)
[Food For Thought - Custom Cursors](#)
[Why Are Delphi EXE's So Big?](#)
[Passing An Event](#)
[Publications Available](#)
[Running From A CD](#)
[Starting Off Minimized](#)
[StatusBar Component](#)
[TDBGrid Bug](#)
[Tips & Tricks](#)
[When Things Go Wrong](#)

[Issue #5 - June 26, 1995](#)

[Connecting To A Database](#)
[Cooking Up Components](#)
[DateEdit Component](#)
[Delphi Power Toolkit](#)
[Easter String Loading](#)
[Font Viewer](#)
[Image Editor Bugs](#)
[Internet Addresses](#)
[Loading A Bitmap](#)
[Object Alignment Bug](#)
[Second Helping - Custom Cursors](#)
[StrToTime Function Bug](#)
[The Aquarium](#)
[Tips & Tricks](#)
[What's New](#)
[When Things Go Wrong](#)

[Issue #6 - July 25, 1995](#)

[A Call For Standards](#)
[Borland Visual Solutions Pack - Review](#)
[Changing a Minimized Applications Title](#)
[Component Create - Review](#)
[Counting Components On A Form](#)
[Cooking Up Components](#)
[Debug Box Component](#)
[Dynamic Connections To A DLL](#)
[Finding A Component By Name](#)
[Something Completely Unrelated - TVHost](#)
[Status Bar Component](#)
[The Loaded Method](#)
[Tips & Tricks](#)
[What's In Print](#)

Issue #7 - August 31, 1995

[ChartFX Article](#)
[Component Cookbook](#)
[Compression Shareware Component](#)
[Corrected DebugBox Source](#)
[Crystal Reports - Review](#)
[DBase On The Fly](#)
[Debug Box Article](#)
[Faster String Loading](#)
[Formula One - Review](#)
[Gupta SQL Windows](#)
[Header Converter](#)
[Light Lib Press Release](#)
[Limiting Form Size](#)
[OLE Amigos!](#)
[Product Announcements](#)
[Product Reviews](#)
[Sending Messages](#)
[Study Group Schedule](#)
[The Beginners Corner](#)
[Tips & Tricks](#)
[Wallpaper](#)
[What's In Print](#)

Issue #8 - October 10, 1995

[Annotating A Help System](#)
[Core Concepts In Delphi](#)
[Creating DLL's](#)
[Delphi Articles Recently Printed](#)
[Delphi Informant Special Offers](#)
[Delphi World Tour](#)
[Getting A List Of All Running Programs](#)
[How To Use Code Examples](#)
[Keyboard Macros in the IDE](#)
[The Beginners Corner](#)
[Tips & Tricks](#)
[Using Delphi To Perform QuickSorts](#)

Issue #9 - November 9, 1995

[Using Integer Fields to Store Multiple Data Elements in Tables](#)
[Core Concepts In Delphi](#)
[Delphi Internet Sites](#)
[Book Review - Developing Windows Apps Using Delphi](#)
[Object Constructors](#)
[QSort Component](#)
[The Component Cookbook](#)
[TSlideBar Component](#)
[TCurrEdit Component](#)
[The Delphi Magazine](#)
[Tips & Tricks](#)
[Using Sample Applications](#)

Issue #10 - December 12, 1995

[A Directory Stack Component](#)
[A Little Help With PChars](#)
[An Extended FileListBox Component](#)
[Application Size & Icon Tip](#)
[DBImage Discussion](#)
[Drag & Drop from File Manager](#)
[Modifying the Resource Gauge in TStatusBar](#)
[Playing Wave Files from a Resource](#)
[Review of Orpheus and ASync Professional](#)
[The Component Cookbook](#)
[Tips & Tricks](#)
[UNDU Readers Choice Awards](#)
[Using Integer Fields to Store Multiple Data Elements in Tables](#)
[Where To Find UNDU](#)

[Issue #11 - January 18th, 1996 \(This Issue\)](#)

[Core Concepts With Delphi - Part I](#)
[Core Concepts With Delphi - Part II](#)
[Dynamic Delegation](#)
[Data-Aware DateEdit Component](#)
[ExtFileListBox Component](#)
[DBExtender Product Announcement](#)
[Dynamic Form Creation](#)
[Finding Run-Time Errors](#)
[Selecting Objects in the Delphi IDE](#)
[The Beginners Corner](#)
[The Delphi Magazine](#)
[Top Ten Tips For Delphi](#)
[The Component Cookbook](#)
[Tips & Tricks](#)
[The UNDU Awards](#)
[Where To Find UNDU](#)

[Return to Front Page](#)

Typically, each issue of the newsletter is posted to three locations. The first is the *Borland Delphi* forum on CompuServe (**GO DELPHI**) in the "Delphi IDE" file section. If you want the issue as soon as it comes out, then this is the place to look. I also put the issue in the *Informant Communications* forum (**GO ICGFORUM**) in the "Delphi Demo/Share" file section at the same time. Lastly, I take the original source material of the issue and package it up and send it off to a gentleman named *Aaron Richardson* who maintains the Delphi Source web site (<http://www.doit.com/delphi/home.html>). He takes these files and converts them to web pages on the site and also posts the Windows *.HLP files on the sites FTP server. If you have questions about UNDU in general, you can contact me at **76416,1373** on CompuServe. If you have questions about his Web version of UNDU, you can contact Aaron at aaron@doit.com.



The Unofficial Newsletter of Delphi Users - Issue #11 - January 18th, 1996

Latest Issue of The Delphi Magazine

Contributed by Bob Swart - INTERNET: DrBob@pi.net

Next out will be Issue 5, January 1996, which is due to be mailed around 29th December. Usually, we mail earlier than this, but we wanted to be sure to avoid getting delayed in the Christmas mail! Amongst the great articles planned for Issue 5 are:

Preview of the new 32 bit Borland Database Engine features coming up in Delphi 2.0.

Under Construction: Bob Swart discusses custom events in his regular column on building components and experts

Delphi Internals: Dave Jewell describes how to detect the CPU type

Surviving Client/Server: Steve Troxell helps you decide whether to use TTable or TQuery

Delphi Efficiency: part 2 of Bob Swart's series on making your Delphi apps go faster!

Using resources in Delphi, including custom cursors, by Dave Bolt

Typecasting: part 3 of Brian Long's series

Building a screen capture utility, which illustrates lots of useful techniques, by Stuart Lunn

Callbacks: part 2 of Brian Long's series on callbacks in Windows and the Borland Database Engine

Review: Eschalon Power Tools (reviewed by Dave Jewell)

Review: Orpheus from TurboPower Software (reviewed by Dave Jewell)

Book Review: "Delphi How To" from the Waite Group (reviewed by Steve Troxell)

In the Delphi Clinic we cover: setting focus in radio button groups, drag & drop on DBGrid cells, fixing Delphi's problems with 256 colour bitmaps and including clipboard functionality in your applications. Plus there's Tips & Tricks as well of course!

You can get a free sample issue of The Delphi Magazine if you send an e-mail message with your name + (postal) address to the editor Chris Frizelle at 70630,717 on CIS or 70630.717@compuserve.com on internet. You can also call or fax him at +44-181-460-0650.

[Return to Front Page](#)



The Unofficial Newsletter of Delphi Users - Issue #11 - January 18th, 1996

```

begin 644 MDI.ZIP
M4$#!;!0"``(`"D1Q^HY43RN0`$$H!```)````05!03D4N1%!27<]+H,P
M$`:@?2!WR*)@6R07<*5)I`4-HJ5[JZD-:Q15^+=JVD1=3>/;QA^;=K*Y`W*
MM>;,@P`"H1,=!`B%K6DZ=ZF:4G*&I$).3.^<X<3/'#3:Z9,&=++H;1KRJ4O?
MNC"-R2VB_I_:G9R7.QIL:7"@P8Z2+24'2GZTJ'4F5&DEB9*,<6JA#07!>$K1
M%:<LFY;N)2JIEBM?ZUH6>2];A8D1>2^6W.?`FLY%:WGQCA?IH.;9_!5#\50
M2P<(J.5$\KD```!*`0`4$#;!0"``(`&YGB!\>**]0?P$`%,#```)````
M05!03D4N1%-+E5)=;X(P%'TGX3_L!Q!#=5^8\, #XB&:B1-QXD(5TM&IG:0FM
M,?[[%4&WD8UD+UQZSSWGGGO;=0@)2PA#_/BF:VZ%H<OVT+57(L@[/?_&LLZ9
MNC;#&UG'%2_KD!_D=S8PA[>Z-L%DNY.V]:!KNK:. *E[B2IZF3)0XE[SZE[1E
M7;4M\RH],JVSMD/)EA68R0A2+"7^DC8003]4SQ`RN,55%TV@S'?=<1OH21WV
M)2=, _HZ[D%+E-]_`? .BY$SY$EUD0H1:P2D#5HWP`U/#J+%,&XW3XUE+I`B+
MO>1ERG`FY&&3%H@P/"BA4(6@KSGI<`M:7#OM)-5>0[BF!36UNF7!T$Q"*
M:] -US$S;&Z?)=.XMDCCU_/AYM8C2N9_%JY<@=6=1[,^]0>3$!C!&CT!]04L$
M?<30F\[*\T8`N/NON4-^WC!,GOG,\]IJ6T_93WDZ-"X;A:JLDVJU[\31<J&
M%RUUS4>DO>=&H+FJ@%>%6:]`Q>O;O#R;;^VZ+^8'\@EO2P<('BBO4'\!``!3
M`P`4$#;!0"``(`"ZI1Q^28D952`$`!H#```)````05!03D4N4D53I5,]
M;X,P$#V@$$F(I5.I>QH[=PM96G3(T_00=FA&I@)EJ*5+,F-5BZ=)`+/R4C!V1
MLC"@"7.^,S$>23KWSL_7>G;P$P_0`8BB'Q?@'@!BPA/!@2LPX<(?(<T`*?N%
MDR6:ZKHF!<T`Q'[A9(DGOQ7;K6K%>YLL4C!\O?YHA1SY?^K"1LI<0)9S6* [.
MZB#0=.1F/S]0H!KJYF5)R' *5'M1GSUG(LS#)Q`*L(.1!2=\VD$!E.M8V`.W>
M>)ZO;$.:9*;O8;A2YX9CO&)=4NHI+XNR'&A0E)6N](3KONLI+[ACW,]<[R9<
MEU4Q^:2!GI7IP&K.J6%N@D"?N.*4[RXX9P\07@*5/'MRD-^0UA'8,5@[@VS`
MP<[Q]BYL\,A>JKT><H+:)X2(^SO$AM^^(AX)R/A"CFMX?5ZNEB]O*W+^;7_U
ME'$<&=SO^#^+U!+!PB28D952`$`!H#```!02P,$%``(`@`SV:( 'QE"('Q1
M#P````,`L```!#3%!314Y$+D1#5>U9#U!4Y[4_'W?W[MW`"Z)%06/28'#2
M$D.(6@TI"R:JA"1/_YIS`:6/P%V<?Y5RJEDEB0.).,L7]TDLY87IHF>3;)
MFTE]J0%2J*U,_V2:S@: ?DWDOOJ9A6FNB+UKCHVQ_WW?O+LON,MI.WKR93DE^
M>^X]WSGG.] \]Y[O?->BPHJ"7_GZ6'ZR/%D<E;*O'?4RE#*2^DG%_08%BC
M)8PXSK;Y;04HJ*D!@/=)%^/^UA87Y: ?XH-ZV;OY^EWL!I)7$5)+TGAJZM`
MGUZ]'@,=PIT^82T'OVN3HGWU)>7*X6L/U]'&Z0TUS:WE;D\M46#95>V67-[E
M#[A;B']M1@CPOU[Q=_G;`HW-?J+Z67LJA!BK,OO=T-V93*`&9EOK/%Z`CXO
M5WQNO(0M8^P\2?5U7E\+)+T[F`YJEEJ+V^)WU?.Q&\H\K$05=C1ZMG>U<F'G
MPT0[N%!\G\R.Z$G>@/.`K\WEKF*(@Q&\@S0;K;%8DDI.(%9RB\(^1Z:Q\X$9W
M<VMQN]L38(H#>E-1X[ [*B.FCQFT$7[";_ZAW2&?G%K9Z`XUU79J--(Q[0IG?
MM["R&0^LLCGJB_R><^ET/WHYBQ<NQ8\9/+PC_1IDK;W3[-5!:#&ACGFE)N
M8.KQE/MG9^;@1HM5NYK-5;#Q)=/'?)Z%6_J2.YR@M1CUNQ@K^E@K/-/Z3FDE
M*YA,4N<'V4^/!&E7S^E)KG6M'%6IG4YB(>D"W>-K);&[V1QQ`SVA`[^O5D
MI$/,$&<Z*1%C6[SM88-E,/"4Y0;IB;<E`BML]D8"J\ :XE=8;3,NSI$Q,:T%D
M](-M;E\XOW@MZ/B-3(A5ODU2)U/<6>-N#31Z/9J%!HQ8:KN!A?6FA6(*XAW:
M5-L<GD$KXS3M9MR'G]SR!F_'QD9/0#/0R9]4^_S/QR_P[@PL*VX;/91W:)6
M"\U(+P8=M<\?Q8'\OL+TPDBW3EV%JD9/+>+P>IN8T@_M9,?<.A#]#C^7[F;Z
M95K]J.*^`PW>6J8\`=U!Q_S0OA@WPZ1D?`&V;7LV;:TLV+RI:;^6I(]0I&(
M:ZDU/RC_]7\ $J8S[?&[^JT2/DV9&;#_EX1UK(]#)Y#3NAE`)(.E?I(X^Z
M+&(67`BHY^;V^IJ<;L5(3\EM@W)[`_XVFI45YDZ<8?;RS)(\C:WOZTYP$(A
M+N1.%QHD8RM7+:BM);'K0GX"9F1ZRC!D.&7XU:QOFYASO&^3D&N^)?AVR]&2
MZ&C8M1`_PZ.1.:GR9>4>Q:=$'*32J80.'$+.S5TV2(8:;TN+RU.+<+B45S5N
MH-85<+E,0L1W$[TB*36NYN8&[$29LA#SV3J5J!2$A;$Q7A5WU!B_J?P`\065
MJ"3,S).$F=DIR_PA=OM<<D3Y324V#S,1'WN,5NPJ7F/$QR+Q2,7[2!'R!^V
MB'*F$4)OFZ`_C42$A>N,$JMV*8+G)?*Z(K$="S,EQIC`^WU,`4TW/&#-/=
MIDA@&7BZ$P66)NOQ@:5&E(=G`W-$A&^+P&R"YX$<-O+`+!'^FE'2^P.N`)\,
ME_&=E%MMWU?#*BC7E0KY4W$SZ(O,X'73$40_88I*[^>O]J9U,E-JP<H-)D@,N
M7[T[@&C"PEZ3FMI)+>( *DYK:./^"*3:PR4A@%TWYFSS=G/4LW^;E/C9Y_*8
MP"SIRE*3NRM3$1+^,C`S\M;06!=)`%S&=\C%YMA(^!TUDD[SH/FD>6(VDA2J
M21B)3<AC(C%*XM(9" 'A;<+WXKS61+S66PHMV98.2]3"#)$%V9(2K`P4<KR
M(VV!@->#I`A[S"8)9P"DY")%C4Q>(\YR+9G6QPGR5)3;8H4BP(<LY!&JW
M1BW6I7D6ZU*BQ0HKGB*ZY(6UVDKDM: ',<L:EQA_L.XN"Y%O);G8BIWQJU
M='S_2K1T7!ZW=&'E>ZS\M>+="E:/`WDW])`UUC-`HSJ^8!WA3FU1CKOG<=R=
MR'%8^24TK)*>&MYOP347\S?ZG3C7W1'7];9U<-L1[7IX'M?#B5R'E>^P2:P8
M>>8\[[NVV6)]#D=\`K.=L)VS78SVR7?U1#ZY/,YGE+*NUNOA4^4B/M73<6[Y
M'=5MN3T/[5:O/>(V#8U*(K>IOL[=9MHEOP/Z/AS;_L4AI+RU2X$4:>9-8:9)
M2+5M3^&ZFSQU7K[N$/ /NK]$>M^Z1B'YM?P_1_-D>E8B^>1+1IT4T9)>D%G\]

```

M_ ' () [Q%_Q&/!M[LKN557)]X\$T85C[A-=-^QV.!U['5&;\)]@^\28_-IM_@W;8
MA1,NY8UBMB-J^Q_3S[?]CT4\ 'W\$[\QAU!* ,]3_C>3JAYVG-<_STIC4GO/L+
MX4\]3:I_#HVF:72Y1K,TNDJC:S6:K]&-&BW3:+5&']9H@T9;--=JIT5Z-]FOT
MR3E1#&OVUT# '(W*"9@GO#A%!\$2CWO`6T6N.YIP)0[B\$/M%\;Q_O>#:#/:_*3
M&OTYZ/V@DQH_I=FY!GH?J,)4>1I3X\CB>=.^X;S\$(I).&DC!987ZR22+#(7A
M3R6K2"F,?"+);U3_3222LJ6R"<1"RE5D4\AM4GA;S2\ '@Z5W&(NW%Q67KRU
M*+NLH)Q[/<1XJWP85]_&2+LNS4`.FRZ)7`;>X;X(<^LX'MD^8)/Q;O(U&\(
M?HF!=WYCX'\N>+N.-UQO@7]'\.DZWKZ\"_ [W@K]=Q[N&"^#_1[O/] ^Y/>`B,
M\VDZL0/#ID7PJ3J^<:6`7R+XI3J^82P'?X?@%MX';\+ _+V:/B^NZ\ \$7:#PO
M?/S[V! ;!+]3QBE3.^)<A]3ZO!WO`US\$M7O!-X/<*?H&.OSX=HK?F?+*/^G[
MP7^#J?F8^SWN/)ZI/G8K6O1,=,MWHC^-06NW%EU4/OH7)[J]S=BD'\2&6(V)
MR85-H@Y%NPD5M14UJPT%I!OO=R_>M,<H55Z(T\YA6F@X3%F&IW%P.H(CU+=@
M_T4<9VZE(<-M=,KP6?J5X8?P]QI=-F227EF!D\@==(^214YE)?F44XAC#)W_
MG?0#Y7/TIO)Y]/_0UQOT74EASZ&>RC3F\$OKC/=2J7\$UNO?5Z-V#B/E=.FQ<
M2\90T##QG7TMGS]31GOHVN`8LJC#-/)]=+?I?<SK`E68JG!5\$"]I@UTQ%1(
M+YB*T*H6H5&]C#E_@N[02<Q<QHO-&RG;O(GRS0_0=G,(N9!9IWDS#9JW/? ,
M6^FDN90FS&8V)*6@H7F0F&4;+;:44[9E.Q5:*JC*4HE.JY(Z+(O9)6DYFHMJ
M.F^90:<M.^F<91=]:-F-YF8W6IL5+\$]W%QH!%SFCM]!#UAHT%(5H+7)9MVX]
M]NDZ&K`6TSO6INS]C=CY\W%+L\$FVD3K;,VTS="C=F#;7DS(WTY.V9KI1.V
MO73:YJ-S-C]=M%6S//T>;\$1ME&EOISQ[!W:W3FJT=V%_<[,^?1,VBF[ZD?VK
MV&/VT7OV`NPRK6Q,W\`2';V4[?@Z.1W[::^CFTWK][,CCL?H><?C-.HX0\$`'
M`>:Q9I!`N17(I&>,)=!.`?1;+FDE^#N!SP,Y0"YP+^GTJT`7`NN`]>#O@^[]
MN,X'-@`%D!6!.H&-L+<)=#.PE7*D4M`'@7+H5()6`SMQO1O4!=0`=4`#;#:"
M-@SM0"O@Q^P/V@;:"=H-V@/:"[H?]#`@,*Z/4)/\+=JD^P[M,!^E[R<=HZ\I
MWZ7W3<=<T7@-<1T\$G/\=^B^#GX,^"GP%O` ;X&W\$UO0=X` !/X+`'O0_6]<
M7P`N`A"]]A'H)\#_PMXTJ,P\BL)R)"-H" I#*=/K/@"X';L/U[:!W`7<#ZX\$\
MAKR!E@;@`*`@`K)*T#V@CX`V@7I`.T"[0/>S\Y1!628/G9:.T)^D9XATSX(
M3E\`P`X?LA\`IX&=`\$`@?N`RS`#,[+2T&5@`Y0#[+,CT`NAFH!MQ`*]`-`!!^
M1I,>`"J`7:3H7L5`.T(GI`GP?P\$6L=&DSPJ]O2EY-)2\G5RI9W%]GI:F3M&Q
MU"OTQH(D)J<%-C:4O)`Y%V2P_0M6"OWJ;7TQ>(NT!SV'TO6,\$-Z(;O36<IV
M.W>Q+Q;7L`YG`YLN]K,WG#WLS<H!,>;I]/W`:`"+/9W^F)#]Z\`^H(7M-`P'
M=>HHZM91U*VCN#Z&VO4,Z#.@ST+^+.TT?!?U[3BNCT/G..K<*ZAIKZ"FO8)Z
MQO\$J:MNKD`^CK@VCK@VCE@W3EY41T!'4N!'4N!'4N!'HG\$%].X/Z=@:UC6,"
M=6X`\DG4N\$G4N\$G4-(ZSJ`5G4>O.HM:=1:T["YT/4.<^0)W[`/6,8PKU;@KU
M;@KU;@KU;@KU;@KU;@JZ`Z/6?8Q:]S%J&\<5U+PKJ'E74/.NH.9=@OYC%TT,
MYU;&M@LDX;R:Q+*!?"`H6U`SK&P0F!"PX61I8R>!H)R.NI>.`U\ZZQ#(P\$SO
M@V4#A4`54`\Y2S4OBP<P+*8W<K!#UXKV3G@0R`\$!.75J`^KF1/H%UB#;]:`
M@>`"@;4P`UL!]"L(4XI!2R"TI!0WHHZN)6M`SH\$2G&8*7U0%#>B5JXDYT`
M+@KLOL>_BYT#@G[]ZF\$]609ZUBO0P/*`<J`1",H^U\$0?VF4?^[\.`G`T:>`\(
MROM%_>AG]W]@*['3V@/<P)!5^U,=^]CP0!A@HT!XOZT8>/_D(G:47W[]
MPB)VVAFB\1Y%KAKOL:P8SY.KQ)T!Y_6!RNE!Y[1Q-'#[H//Z8.7T>()N]Q8!^
MQ4!%TJ#3,EAI6S%0#\$%&#`/LCZX1HF55AX:OSRR:=ZAO7'?HP/69)WXS<.`/
M,P/?O#!S^L#%&1T<#EQ[XB<#!RY#]C%D5V=TAX;_@,&/LI=9&9K"UW\$8*1,6
M-'0&3KS\$XN(7B`5)`4EU.X(M=O&>TQRU:&:>XNJT#4JHP\$(9+DJU#X3:I\
M/73MS(OZS!0>9@JU+PJUIX[W./BPJ^%A2V`JU&X)M9NT\=.A]NNAAZ[V.T/<
M^43*33FWA]IQ1X[S`CTN#0& ,]Z3,&9>NFE,#B+.1&F=C6:@]8[PG;8Z-V]6)
MA=I3(L8..DUEL>E8\G\24%1*(S;25`/B^3KHE,NB#,1GD<]WF.;8R!USH)\$
M!CMN:@9+U1D<=%K*!>K^SCDLGDUDG(5/>1*?^N"%\ZS>I^P.;T[:>\$_JW,='
MY#'\@L3:6'13`2Q(-81)]S9'O8>1MSDY\40\=U>0R-\M5G&H7)[IMVK'_.4>
M;V!YG;?-4[L.4OI`_QM]F?`V:715ISK;J6*?/[/9%7@5X%_E,SY_"O&\$)E[
M.?T^F4?YMX*7</\3G[_%AE? (;X%W/) //H;_ *U!+!P@90B",40\` `` `C` `!0
M2P,\$%`(`@` ;J1`'S:3%;O3!@`FCD` `L` `!#3%!314Y\$+E!!4^U:W6[;
M-A2^#Y!W., "PU084-\50#`60BRQ)VZ!+FM59FV\$8#%JB; ;:2*)!4,J/84_4!
MMHN^Q/8PPPY_)%&)XCA-:\>I!72Q)) [#<[[O.R1%[OWZVOK:PX?P[T?X)64*
M/M[^LLX`_`PDV4Q"XEB/.U"OS^B*14L[]/J<;\/,UP?"H=[5(:`9=AH\$,J
M0'%(N:(2!C3FY\$#;53H\$>,IB"D<DH5T(XZRW?[37.=[IP0TOSR%>.[D:<]&%
M`3EF*?0Z;UA"8O*6TG-R4X=[1%\$(!<4_41<>;6YL_K#QY'&]K6F39Y%M[U]S
MH\+A"4L\AYN/NH\WNX^?9(G?UK2I`#8W\E+Y=E\$L-%8=>\$6EW7X]\`?/=OWC
MI`VD]=%&DS&3D/`H1*)4C3)E-3RH4D>:P0EQZSY\$-28@M8!R'RP(2<26QHW
M[AHB+=E`P6`3UR\ [B! :V'W\$2`S/N9)YE7"B0-(U:[<#\W8V)E*VV[\92<)!*
M1=*OMMI`56B<H1?C+2\$34.2=BTMI2M`["<>,GE'?TYB?XQ,10)XJ%NOP4Y.#
M[OCT]!1=)Q3U&\$F08Y['\$=#4#"#<\%8"U*WQEV,4!]\%2RB`IL-T'(!,5H
M6*KPG[;\$;!%2('\$,V'&JG')ODXS6G!\$T=#0"%H,+6W11H,.TX.-,\PX.J="

M(EHN&3A`P]&(2N7[PXB!1)'.+H\$Q1>\V90)N:'&@PS!/OSU(:\$HP#7:&W2ON
MNQ(TBTE(#5"8`C.^F.IHR?RI Z/OS(B`'K?T`Q/B\$&W,6XD"0&WDV+T,L!WR
M2_4/GBK!):,A%PG^2:B49*1?G;/T1`-DG*VOO8>-Y@MZJ(B(B,B"*P&'\$.<?
M05--9F`BS@0/:90+R_P\$JGI*'.SEM4;4HJ<O1R\M0!@/?#!6QHJC&=Z75I+
M>8A<Z7\$2+:42+!W-:J@\$\$%U:0V@;#\$J23+S/:9SMZX1;L-AX0YXD2(#I\IR+
M:\$S8'"Q)\$6G,TQ%2=:WE&='0A%A(8\Q>6PXXCRE)H5W]WKK,+58?&TXJM!;'
M+09#1:U+1^XT@>:(O'U<[KF]@+?Sv2T)W@IFKL\$PJR6.`+C&/0)EJ<!_%JD
MJ0>F\$0;0WFH&Z26.IRN0+\$@SCUEZ0>#ZU!CV)(,9ZYF\$(<V4-2[KN8&;?4M/
M6<9+R(TB8D35EQ3P"SKQ9X7E`LGJX1V=E!/@#284_'88JB+'GKXQ(KP*I6-<
MDLJE+//+*(5CTJR&0XX++T\2U@R@UPIO>8QEB:?'`V3VZGB,U>=B>N0GQ58
M+^.`T\86`NGIA8Y;\)W^I(+)<FECOC-NLJ[9C?GR:KR8`>VZWEB:?'';LDR:M
MF?<YU2X(7X)\PQW\$UYG<>V, X=>&2`ORUM6^^5)%VB13F.R!U%, [\0'W:?\$
M&O&4%L%.X[4WYN?/L<07./W:@,<81,]\;]_0\$!6LD`SO4&7!^F0FRR?%S>-
MDXK="EGD>M>&G,A1J>\$30)DU4Y*BI*,&-1W,MZP-\$(<)/YN80.XR], "[I8:
M[F;ZSL(;39/>DUM<T4Z+W<W`QK?V(=?>O>H?'+W>^>E@KW_8>]:O]IBVX9M'
MWS\&Z_PK7R@;PJ?5<X+>=IU-5OG\$^69#%-4)KF7,+,\W:_%APA0DC\189
MMS]8<=Z75!6@7<1K%L0N(C5#\5J,L%\$>VIZ-DNVO.L'84CO)=+.*!+XTNVV
MXL!';%40\9OASRVWV';,"2QM.LDSVK);-6Q?1!\A%BVO*#;QH`-H06(!.NE
M-&J5YFT,2F]\VW3*7FL9=\$(>4=U/:;5UN8W99'.QN-=5Z`7@VX`6Q4(/-:U_
M4\$S)M2<,) ^IRZKN8??BG"0Y)E]04P&\F:[/]FF*:097Q[R;E8G.XT\$H^B%GH
MA"*=4K[VG5XG05^MMJCU3U^%1CU>->D4/,0#WT5=4B<!GZCEHG)VJU[3S[
MMO,<B'/5BS5[4C%19]`!\$YA4@RKL.JM+MD\$^CYKP0;AO%#:J.H*K`?H%[-?/
MOP=UX)N4\$+C,`AOGM,I:'2C<G0.%>8FGXOMZ\00VU: (>IJ65@<@`QH1:L`W
M<VH3FV%`6!W0-!W0S(/%\$OEF!C&KP,5X#7_W\NAH3@QXV%U)PU3\5T=:7^:D
M9A[\>]PUDV^1<64XPV"Z.G=-)L5^%>L=:YD<G4L>*MCP7DP[&'8S"ZIXKFJ
M1._Q<>7<*/#@: ^:AC`#JY\`-]/DV=RV=`<!F*O:G<G"/#WGGP4`%7S/X)K9I
M^*].GIM.GN=71WY.GT.C*!(+JAB)<G\`H['Y[]T\ [&K<X\$)!&5LTTZ"ENG8
M?KXG,QXR%[#USN6O&J'N_O]1,`^!5BC,@B#^ZJRO_0]02P<(-I,5N),&``":
M.0`4\$!#!!0"``(`,]FB!/_UKKOO@0````)````````1E)-0TA,1\$N1\$-5
MS55?:%M5&/_O3=_F]2NJQ+7+DGMBSFQLDF'0P9)3.D?6"68EG:@I;&Y6U)O
MDYA\$7;&,/A06P8<]`#DTCU8BR]E#X(BHI=11,>4^K3J@TAG<4ZH>Y'Z(,S?
M^[\^;5817_2\$<W_W+[S.^=WSW?EX'8^#%"VZ9I&M;RVB7\+N-WW/:>9V
MT7:=/A%\$'Z)KU*[Z6EW*2=J@?][.HZOJLJB.IX1;%\$3CC)'*Z(;83[_W^M>
M%1=\$O]C?MHT>\$_.LUJ\UYIC`72QNI*-%G[^]!RZ0],5\LZ7-\$<UN\$(+G@1HO
M98PB4>3W@-BSR(E,=FP^KX,\-]V/XU?)>"\$W`_701\$1L,A1O5A,GN693\>G
M<#PF'3\$C62PR]^M608Q8\$X<*R7PZP^J=[RZ(*8N,Y;*E0HXW?W=C&4%BTC:8
M*\R!^7)P'0[+]08R22-W%MSM;Z^*94N:*5BI0)+7[I-M,ZDXAZ37F?@]L]B
MEUSDZM-DD%W*+OQWC6HR.-;(2:IC2;"U3PDK/[60B@5P1+?N6B\$ (9_-M+D
ME4WEM%(/MMIDO&"I/1L*5.:+R\(%ZS7/%)A)=4)RS.YDEXL+X+.2S3JL:"[V
M*6;DBOJ!X1;L_ [9*Y/#R;BMR787LBA`B>VA5V`#M6N9>S@QF##V1?6/&9F9
M%P\,^Z#ZH:.;E"IAJ2!H;1!\$BQ5)\$)*'M*K\$:C6)ZJJ%2NY/Q!'[[Z^NG3Z&
MEU`K:WBJ]H2>3>F%SF;*XX1<B:C,Z5,+MOMEB2`#*26T(TS4;NDV!&[:]&T
MW<9EW#+:E66JBI5[%>UH'/:80//BLV*@MM=-.0@0?ELJ3CEHXX*&\#3+=*"
MYW7"SNBK#N)MP>`8:"?N-Y8K8-JF:I10X9B4,],:U#)2`S3JF:A1/0-AJ&<>
M!O6,:Z-JIG50/<,\)U!9;50K(SA:]IVA@&?PV='8`*F!Z./Q:,(*&=6Y@<%1
M)OKD9V"``J22VZUQC+NPS1\$>MFH<P!!L/=9P&[?RAG!36&FBTXH'8B^`RD'\$
MN0V7T8>QG]94+RX&\UVTJ[(M1\$&MC;)>&V6=702#Z!@[>A=Z"%<+/?>?
MB#TG#WG1_ ;3Z1%RAWNZ:4CIIOHAWMZ\=Z+]UYI^`C3A.[!67WH?ES2(+`+
M/41N>A#95K[O:BN)^I^UA@JI6K6P5IGD];V`#\`Y3W%14#6X%QC(5S\$N%;^
M+,<#=#M53W"``OZ:QE_?.6.^Q?3&>;M)OKY<<U#[:;S74\&9I&'HJ">"KF5(Z
MDPW*=8+1NN93E.)6656FZ59JFB=Y&=M,%K\EW(BW9S&\=\`HX8G0-N^A>
M6\!%X"V5,UTW[P"/'\`#+%%;D"5C0!^P'Q@"7@0>U=AOW8QHG/.Z.OR,``">!
MT[P?`!0PKW\$RZN;WP\$LT:8:0825@&E@&7@;&@5>`D\"(@^A-8!PX`#2!*>`6
M,`V\`S2`_+^69ST2[S&V`Y^CL+F%=8X"6S!O!@77+`"TO\0D1FR<:TB,VCC
M6@#_.-NI"!&TC"49UU!3RI)_#XBY)`&0:^%U;^)^B`*HB1UZS32YP(MRQ"50
M0METRO5T\QNK<II7!/\!(-Y:??P[]`#_BCEY/G[*WPW^\$[PO17^R=H=P-0
M2P<(S):Z[X\$`#@"0`4\$!#!!0"````(\`RH1Q)^HI[5S@\$\$``.X"``````
M1E)-0TA,1\$N1\$9-A9!-;MLP\$(59N;9^#`/=>C>[[(SLLW+D&C%@)4(DU&M*
M&DES*%(@Z=H&>I1>())N>IJLN>X4<H*49.S50H`4(<DC->WK?_(I(OGQ,XK05
M>C\$GU],?'B%YNKR.\EIU<<MX-0`?JO=KK,W@V1ODLA)\)\,-JTP[2-Z-[I`U
MK1E\ (WY,>.\D&/T114LIS"R67"K?+_DMI^73V+V]JKR7T-WN:8>C29)!1H6&
M#!6K7Z69.7`<\$]NE.E?[4:V3Q<K93U*V1ZYZ35"M1MMX^2*5FQP3^M)<+K.F6
M&_OT(/C!_0T*SB&_H.(N=3H.TM71CGNS2G/E(SR-2V0CWG!5Q4*P\S!H70D

M2.Z=P3^>P;V;X9RS10A#RFU#Z"Z=%?J!H;\$]4;T-9F*3@XL.5_.K_P[G\5P
M/B34;!6%)(>L5*PW8\$V9H5Q?#FI4ZUO)*Q*E5-E?'[\\$Y(P46*1[:5!?\MS\
MQ9.>>`+*8\ZLRS^1OMJ>@\@F0F185V`TT_8Q?;KL!U2!=EP8C02-"*W=0L;K&
M8SHHCY.P%;145!Q/Z@ZA0ZUI@S.`O&4:["I0&\`] [9C`"G:M%6F#?<?] \$8V5*
M;IL6=C:#\ZBPV#8-JEFPD:K:*-J'A)#?4\$L'""'ZBGM7.`0``[@(``%!+`P04
M`@`" `!NI\$<?]76VWX8!``#`,`P`#` `` ` \$9234-(3\$1!+E!!4Z5236L",1"
M" _z'.134(M*STL.R8A'Z1;7T4'K(;D9-.YM=,EF+E/[W3K)4BU3IQVEG7MY[
M\S*;VAH/"U>D*])J-UJMXSUZ!8JQ]#4C-QN`<PV?.\-<1\>C)UO*FRJ6U?F
M4ETALUH&+"7%'(H+IZJ5"8=I:;TK11I\)\J4K!!L;1>52BIG7J7?<`; (7WT":
MQSQ&`L\$YY, &Q.P_ "7C@\\$H(RF&JTW?C.\$^:7*D\$;;D^O2(^!E:1\$73N,XU,J
M&;LSM!J=\$&^R8R\`S)"-;*09) [4UH!(Z?I>M&D<F:M/#9^;W#;M*!1XCD5>`SO
MD5AG9/(M+W:'.+Q"O9^PJ,W\$\$, [4&E.1ONP' [8V.*!(^HA\$;EDN&MKMCH=?
M]MT\@*(B+&3!,7!`WD[NX'OPGER]AVXW>B<<_`&S&2Z-#7\$8:3%P2*8!`?\
M^^VLG^QH:UPT;U/3L@N=(!E"KHA0PZOQ*V,AfD+2Z4/AIW8A]XCW[L-CD=V\
M//7AK/>+0\$=^P>% (D/#_4LEWT&Y)`%!+!PCU=;;?A@S`` ,P#``!02P,\$%``(
M`@`SV:(');\8I/"!``X`D``P``!&4DU#2\$Q\$0BY\$0U7-55]H6U48_\Z]
M-VF2IK7KJL2U2V[MBFFQLDF'OP9)FY*VL\$HP+>U`2[OF;DF]36(2=<4R^E!8
M!`&^#`DTCU8BR]E;Q,108P^Z)C2/:WZ(-)9G!/J7J0^"/WG9M_VYR(+WK"
MN;[_[<[YG3]ON@['Y&Q(X2V35,TI&6UB_A=PN^HXQ7'C.."XSI]+H@^0=>H
M5?4UNY7CM\$`_O)V33U4MVY/"(W*B=L9PQ;HI'M;O/6+=J^*\Z!6/VG6;`F*]
M,NIP17-S,TDS<8KHW1M[T#+IC,_G"\8<46^C!X3@>:#&"BDS3Q3^(R#V;'(\
ME1Z=SQH@ST[UXOAE,I;+S(#=#6]8!&QRQ,CGI_PS/[8](['9%W\$G,[GF?MM
M*R>[8F#N>EL,L7JG>_/BTF;C&32A5R&-_)H8QE.8M(1S>3FP'P=7<>%Y7H#
MJ6DS<P;<G>^NBF5;&B\D(H4<2U^[0[3.I.(9/9V;BR139J+_%[+;G+W:-+%
M;F47]W>/:- (YMN4BM6Y)\&B/\$E)^;ET5""^" (;M^S77A,/AMI1<DJF\I)I>IJ
MM=X\90XGC'0A59@+@@W1J]YO\%*J@LC+V8*1KZX"#HKZ9"BN?E.\$3.3-8-
M-6`_#Q`<=0V\XVI<5R&G(H1('U@5#H1=T)SKJ6C*-.+3;Q@1,S7SZKXA'U0_
M5E2;4B5L%03--8*^?SFB0_*\$5I;8K219%JJ[XBJY/Q%[+\ /72=]AMUOJ&MX
MJLZXD4X8N?9&RN*\$?'(F^V8*J4RZTR-)=AE(+6Z8I_N<DN*+. .V+:+NURWBD
MMTO+E!4K]RF.:KIVGV+S8<5F2<'M'II)<%T^FTI7\E%;BOW@Z1%)P?/:,<[H
M^T[CK?`@`&G[C>V*V-*IFJ44V&PJAFIFV4,A)&-1,UJF8@!JJ9!Z.:<2U4
MSKOVJF:8EVHRJX4V901'2W\X&/!&7QJ)#)T8Z'\VUA>W7495;B`ZPD2/_!OX
M0@%2R>/1V,<=V.80F\T:.S"(L2[;W\$94WA0>"BGU=%+QOMR`?V4__-R"4/3!
M]M.:VH#`8+Z#=#E4>"Y*NM5"ZP4%I5P/Z?G38+C]Z!WH08>7\FUY'/V'/B0,-
MZ'Y:24Z7W,&N3A14.ND6^L&N;KQWX[U;CGWJK4?WXJP^=#^5`=VH`?)0X\C
MVXH!/:2H/]9JZF0=I)7*Y,,WP<+#Y-_45P45(P%7*ZV\$["KIO^^)U=-N5
M3W",:CY-8V_O'+>9WKCG-,Y+M^ANN:E5M[K!7UFVC2-A/YFJI!,I76YCMY?
MU7R!CP@J\QJLS1+):V+_S5^+_4DYD6%,X_CO^+^&CL\!-X")P"[@(O^UR
MIAO67>`AX._)6`3LF04Z`/V`H`/`"##&M;L,(:Y[QA#0`#P`G@%.\`/`,`
M:IR,AO4#`\")-6\$%D6`&8!!!EX`QX&7@!#!<1_0>`,`8<`%K`!``+F`3>9I`
M_JYE68_`\$>X;'@2]3R-K".H>!39@W!(P)KE@A?>@\D15T<*TB2W=P+<#].-NI
M&!`TC"49UU!3BI+`&(BY=`7(M?"Z'+^%0&4Q`Z]95E<X\$4Q[]):14+1<<CW#
M^M:NG-9EP1\`^NKVD]!_R3&GZ:./')OB>\\$WPZ^N\O_7XGQ/P%02P<(UOQB
MD\(\$`#@"0`4\$!#!!"`"/RH1Q]+-W1RSP\$``.X"```,`1E)-0TA,
M1\$(N1\$9-A9#!CM,P\$(:)":W35)5ZY3:WO55[WAM-J;92LUM(1,]N,DFL=>S(
M=FDK<>!>`\$N/!!`7H\$``,`?;72HA@6398V?^/_\WOV*2+)^GR=UJO9B3F^GW
M@)!L[R)\TJW2<-%.1^]5*_66-GP:Q#FJ@L_7PVVO+1-F%X-[Y#7C0V_9JP
MSG(EAW)\$\5).TN44)K20LP%*Q['_NU)%?P<^=L]:W\$X23/(F#20H>;5DS2S
M)X%CXKITZVL:5R9=K+S]9,./,*P&]4H637",-LKP/@&=FJ!%=L+ZYX>I#C1
M#]SPG<`1?9")4:IM_1EG./1GO-,R3!`LQV*L=B)58G2<GORZ`'IRO-R!G_
M#![<#MX(7DLZ8,(UC/RE=4(:69:X\$7+8"8N.?CH<#V_N]P?EP,YW7*[%XS
M2/"LT["\Z46R; ,Y:"&E9DK49)XP[3[=?\E(L)(D4.Z5Q;-)<_M7SSOSCP1
M\$XG@SN6?2%]<3_\$(R&V#&MP&AGW\$3YG;@!E0OLN`56`0H5\$`*'E589`.BGX2
MKH*&R5+@6=TBM&@,JW\$&D#?<@%L[-!;PR%HNL81#XT3&8M=Q63N95ONZ@8/+
MX#U*W.WK&O4LVB=;C7K1H2OWU!+!PA+-W1RSP\$``.X"```,`1E)-02P,\$%``(`@`
M;J1''\A^II>)`0``S`,`P``!&4DU#2\$Q\$0BY005.E4DU+`S\$001?Z`^8@
MV\$19/+X:%<J@E_8B@?QD-U,=Z.3[+])5DKI?S?)8MB%BQ^GG7EY[\W+;!JC
M`\$RMSDN2V:C;Z7:4<6@7(L?0-(S<[0#,5OS@%/`\$`'I69KVILJSM;Y;Z1F91
M!"PEP1R*"ROJ4H7#M#+.5EX:*:5U1X[5X*JPA<S)U-GB>-DYWT#;:ZP.BT5
MR0F<01X<>_,@[(=#`,`KH4J)QRJV&,+2&=)H>W)3.>0]N/8I4386X_B4*L;>
M#(U\$ZXFVW00F;@1+86<U.49#T9.V_6C26W54CAL_=9PU[8@T<>S(O`8-I'8
M9*3R+2]VW]&X1+F?4#=#J@AG8HFIE[[N!^V/#BC&?`\$#CL;AE?]'0;G<_+3O
M]@`HFE# [!<?`5D?W<-)<CZ]WH1N-WHG3/ZXV0P+94 (<1EHD%&D%H<_PGXY

MZR<[VAKK]FU**GIP'"1#R`412GA3KE0&HBE,C@>@W:59`'O\$>P_@26>WK\#\#
M..W [M"!7 !)]!CS U+Y;]+MO`-02P<(R'ZFEXD!``#,'P`4\$!#!!0`"``(
M`,)FB!^\$KY/#P00`..)````,`1E)-0TA,1\$,N1\$-5S55?:%M5&/_.03=
MF]2NJQ+7KDGMBFFQLDF'0P9)34E36"68EG:@I3&Y6U)ODYA\$7;&,/A06P8<]
M^#"DTCU8BR]U;XJ(7D8?=\$RI3ZL^B"06YX2Z%ZD/XOQ]YZ9)5IV(+WK"N;_[
M_;[S.W^^>[X00Y'Q8X2V1=,4TPK:)?PNXW?<]KPM9;MHNTZ?"*(/T35J5WVM
M+N4DK=,;^?E<TFHJF5/";<HBL81(PW6#;:%OWN?>:^*"V)`W']=OUBK>6W.
M:'\$VE3'2*:+W?]%^EDE[8JY4UF>)!IK=(`2/`S5>SAHEHO!O?K%+8@OD1#8W
M-E?009Z;'L#VY4B0\6(^!?!+GW1'AM\A1051*GN613\>GL#TF'1\$C62HQ]\MF
M48Q8`X>+R4(FR^KM;R^(*8N,Y'/E8IX7?W=)"4%BTA;-%V?!?!%=PX'E?\$/9
MI)\$`^([V-U?%DB5-E..1<I&E+)]TF6F-2<8^=*<Y&,EDC'?E)[]"+7/V:#+!+
MV<`Y7:..#(YE.4EU+-KV]BLAY<?V%3\$/CNC672N\$)^2SF9:5@K*AG%;J@5:;
MC!>,D;2>*V?<+Y5YX8+WFN=+S*0ZX7DF7]9+E070!4F'%,W%9XH8^9)^(-:"
M]=_&U7!X>;5E.:]"=D4(D3NT(FRX=BVS+V>C64-/)%)_1(T8V]>*!F`^J[VNJ
M#:D2E@J"U@;!8*DJ"4#RD+8GL5I-HKIJH9+K\$W'\$_ONK:Z>/L?97ZBJ>JCVA
MY])ZL:N9"M@A[YS]P50YF\`UN"7)(0.I)73CS*!=4GPON_6AM)W&:=PRVM5I
M]A3+]RB.:P'M'L7&GQ4;506WNV6Q@/RV5(]D(\ZJJP?3[?(![7!3^CKSKN
M)-X>(`G<3UQFH=5,M4C1HR%\$8],RVCFI\$PZIFH43T#X:AG'HQZQK717J9U
M4#W#/-2066U4*R/86NZ=8;\G^NQH)'9J*/)X?#!AA8SJW%!TE(E^1GX0'Y2
MR>W6.,;=6.8(FZT:!'S`(7Z]E;N%6WA!N"BE=-%KQ0.S%5SF(.+>1JOI@=)*J
MZL7%8+Z;=E3V!2F@M5'.:Z.<TXM^\$!VVLQ.]&SV(^F[_F^Z@'[#FY"\$O>B>M
M/!&W.'>'AI6-N@F^N">/KSWX;U/^C[R-*%L%<?>B<N:0#8C1XD-SV(;*OL
MN]:+@OYGK:%"6BE>KTSR^NXO/\$\$^17%14#`F<;C&0K@`NU;^K(,[&I;V\)\$
M:OAK&G])^YCYM/KY^WV"?GV^34/M?=-:3P522</OTX%7L^5,-A>0\POB=<VG
M*,\3,\J,-D,S5-4ZJ9.U@<'2OY03Z>8TMO\&#\\$/Z!QP`S@/W`ON`&^IG.FZ
M>0=X!/@K<!'8@BP9`_J`\\`@`"+PJ,;GULVPQCFOFS%&#&)G.;U@*>!8V3
M43>_`UZZB23.(#`L#,\`*#\#(P#KP"G`2&'410`N/\`(:`)\3`,`W@1G@':`!Y/^U
M`NN1>(^Q`_@<A<Q-S',4V()Q,6!<<,4*R?,`B<R@C6L5F0\$;UP*<C[.=*F%!
M2YB2<14UI2+Y]X`82Q\`N19>E_Z;N"B,FMBFUTR3"[RHA%WR)E1,IYQ/-[AV
M*J=y1?`?.*MU>U'H'\8_D<I+<4.5[P'>![ZOR3];N^!]02P<(A*^3P\\$\$
M`#e"0`4\$!#!!0`"``(`/RH1Q^TD]1%\$e\$`X"```,`1E)-0TA,1\$,N
M1\$9-A9!;!MLP\$\$59N39E&0:\[6YVV;E99]?(-6+`2M1*J->T-)*(4*1`TK4-
MY"@]01<]4)>Y0@[04HR3&BC0`@0YI.9__3>_(I(O/R?QS6J]B,GE[&=`2)XN
M+Z.\TFW<<%`&X]J[1HK._@>#'+5#=#X'PPTO;3-(WHQND->-'?P@-&:=Y4J.
M_HBBI9)V'BNA-*6%N!:LN)_XMV=5\#3VMUO6XFB:9)Q:2!#S:MG:6:/`B?`\$
M=>G6US2J3+)8>?MIR@`H3(I_Z)8LF.(2I,KQ/0&>=6F#%=L*ZISLICO0+-WPK
M<\$S09"R40>HM?1GE>+G/#,RRM=LBV(BMF)5HK3<'CUZ0'KR@)S`/[Z`!U?#
M#X+7D@Z9<UC?VF=D(:6Q>Y\$`3J8J4L./CI<Q!?'_<[CV7#>)<SN-(,DAZS0
MO+/@3+EEPPIP;E29:R5*\$J5,NU_W7T+R@A0ZI%MET9SS7/W%\^G\$\$S(1"AY<
M_HGTS?44]X#<-JC!;6#85WS(W`;,@/)=!JP"@PB-VD/)JPK[=##TDW`5-\$R6
M`D_J%J%8UB-<X"\X0;<VJ*Q`?6<HDE[!LG,A:[CLO:R;3:U0WL70;O4>)V
M5]>HY^%&Z7*C63<FA/P&4\$!"+23U\$7.`0`\"@(`%`!+`P04`@`"!NI\$<?
M%_#LRX<!`#,'P`#`\$9234-(3\$1#+E!!4Z5236L",1")_Z'.134(M*S
MTH.L6(1^42T)E!ZRR>BFG6273-8BI?^]299JD2K)..W,RWMO7F936^UAZHPL
M2,E1N]5N:>O1+87\$V-2,W&X!S#=#[S5Q'QZT76PJ;*I;5!I072&S6\$4L(\<\$
M<MBPLGJD+PZRTWI5!&GVFI3,!FVA!Y2H4<Z\R[XC39!]\(VFQ="8K-*D,SD%&
MQ^XB"GOQ\$(!RFBFT700-\$!:7(D<.;4^N2X^!U<A):K:81J?4<G8G:-5Z`+Q
M)G]&Z4>P%@[&TNO2!C!QFJZ73"JGU)\CX_<AMTT+"D,\)R*/X3T1ZYRTW/)2
M=XC&! :K]A*;64TTX%VO,@0LE/VAO=\$0QYB.:@*4MAXO&=KOCX9=]-P`5(0F
M+@%CLC;R1V<#B;3J`?8[4;OA(,;C;'E;8Q#B,M!PX)!6/\`/\-^.`LG.]H:
MF^9M*EIUH1,E0Y"!"!6\;E]H"D4LDX?C)_99;A'NG<?'DU^`_+4A[/>+P(=
M^06'(\&8_Y<J?`?MU@=02P<(%)#LRX<!`#,'P`4\$!#!!0`"``(`,)FB!]%
M8Q^`*PT`.\`<`))`))`341)3D4N1\$-5S5A];%3'\$9]W[VG;6+S:8B!HP2#
MP5!L""5J`OXV!.R8&&,6S)G^X\$/SG?6W1D,!6(21YPK*,3U'R:UC(GR0164
M(\$\$JITK*2T6H`RF(D\$I34;5I2=(V*`&5EJA*[/YFW[N[=SXG2O](U8/Q[, [L
M;V??[NLSI84U^01?A_0#CJ8\F7*P=3NU&/XMRNM)^U\$6NN4;GFNC:1AF4BA
M?]@SI^9:LJ2#,GWKWV'Q]Y1T73IJJYU*]9)<"DKE%64+MWU(B_O.D'C\T6G1+
M>4;IDI1L-TMZ/B:6MM:O)6E1/=]^CG=Y&%8;=7[0V&UC6CU%#O:X;_5`5%-
MV.L+\$15\F25]K@MKO?ZM^]M5"#MWY&'L46%5,-`,X:=WRJ0L75BAAD*>7=RR
MJ*H>8V.AO=CG"858]L\;`:E,;U@>]+2W>AE]ZP_=4KTN+`XP\\$`&W_V\BG,
M\$`NM98%&R3OE#TO=>O]E7@]OL`NR/[V_B7IE`XM[0P7AX,,?7SH.CY;0"M4
M?P<D(P)*%V2I`\`+>H(AP-^R#XZ?MI^G5M97%O%Q&PK*2J98OD[.<FY7!\$K
MD(L:D;-"\$?.GUQPDVV)(K*VR-^G^>O=3TD'(B#X>UV=YK?@[A=ZQ?:7DV7MM

M] ?8<VWO*"70A.]O\;; J9R\$ \$IUW+ \D3Z3?0CV] LZ0&5>GQKI@02D19&%-, V0
M5JK [*HH+ (]W0G1. Z) DNBKB@2@6XDBIL"75Q9 '#D&Y2VA/&F14] O\OE>MMJ_
MBYR\$BF0#EV+T6>W9JT; ZH) DC-+?BUEA3& (KT0 [= &Z, XE=I@?&8! JBU"MD6, =
MEG9ZPY\$A: /9\$3?3' ;U!] [9&G (3TBI"/<E2XM; `ITA"-GH1H0JCFRG-; N] U6'
M/>&. T&: 07XV<@VY8?GSN6] (1679!MS40\!5Y@I'S4%P3"@ (HU-) DFL"+T-V6
MC\FC8G+~NJ+ (, '1.A74# \$W#%D5>@6R1T> !-ADY, D@; ->J&Y%D<9DW09NGJA
MVR) CX3N\V"PM@7V1\$<A#"G_6`7UB=?E67OZWH3LA=. L5. 3VF*_ : \$FCTM: N0J
MU" (=; TBSXBI"X-!CW^7NK\$97AVYCC; O*?HT*R[>., 5!U1-6, S: DPW_#<%1 [
M&GOK5*E) ;) 9QDAR*HI3- /4%0FB9B9; /QEKF6KFE16_IG_.49, 48IB; X9K' /
MV [PG8X, ; N-LQ7*/59&%83H04&9`E@, QW1B&] 9@F: 5JB) QN8E<!LB6&NF#&-
M5B7=Y'0&8" T' \4`= \R`7JN28?8Z`U\$`Q (NN*"+"9D (LTC\^OFH&9`, @LV. 0
M!C/DMJS, F+B8! JH*J\$TQ5*\9=<6JS) ETCOUH': "'8M!WS-`&6VP: V!L-P`X`
M`HM-PY@9D&N-+8_NO@: D%9"78I` \NPDR9BMQ-E: W!O9M\ /K#&8YVM/PB- IIZ
M<\L [UI*4QIKV%G@A1^, , 1R?: +DV) MM5_1EL1) *. Q6+@=0BG" [BG]: F_VM5>K
M_A: B\+P+=G' Z61T [X; 2MOI9" ^, '%\$?MINUE8A`F] =M-^P8Y#, "8L) GIQ#. >!
MW3B1^0CXWYW (%^PC) IOVZ (DL5H! ^ZCB, : 9!M_ &UJ< (&+WL/6Y, F`4*E6?3L+
M; 4+\$TV"CU] 'V?<=Y, V**V, P&PE; 8' /8_ (M<0LC; U=3-5 [%N^)?JO) -HF' ?Z
M! , , LBB/6.7.<"8C&9\$1C`J+=69^ (Z\$U&]"8@GG (>3T2P9T] `L"B.N.Z\E (@8
M2T: ;) 2! LKM%\$Q) 7D45U) &-5: UT) 7`B (GV49. @HW=KII\$1\$, RHB\$!, >`ZFHCH
M34; TFA! <KWK>CD1<2?Y. ^X8WP&1 (X2SH, 4; #G\$%) `J-&: 9GI5A2N*-0M"/>
MZA, Z8E'4=\$/*QI35B8CZ9\$2] /68ZHPGQ. !2N: /\$6MWI] +4'5+XV/L_Z@-"9&
M, (Z?OBO<XF^ZL6, RR2PE6F+PE0; ?8/! 6@U<90, [@_RAZ*\L00 ("FZ8YIR#0%
MWVIY1&+ [\V"7>: 5QBUL! ^SR2/-3=!O^P=>!_Q#2"O' 'P!O\$: --IC\$/&+S'
MP/WB*S+I. ?!5*/T"/ (CO>%T/?^*GQ\$85NY@K9+J0HQ*_B.L5XP*. 2OSBK5#\
MP@U% **~2OR"': ; HQ3J=XA?J=#)=I#=3] `+MIOC%^2Q%\$XE4BD=E! YFB, 2IE
MP; 9F5) K, E6: .24^7SW=6E&RL+%U155BM^Q\9@I (RGDUZ1@0\CO^E&+) KP7R%
M () <FX@&A/M.E<%RJ07T; '-FU4. &@LYUCN% [EB- (H%E)] 4. 5RT8/U5KF8I' `N\
M< (/=>I4WNA^-`UQU*; R+>8. \$] 2IO4?: 1_7J5]] \AD<Z) *F^N1U&-<'6JPEND
MET, ZJG>Y%?90706?80: F3* =I^\$47&' FFA! /%14W8>`<LJ732DD; G+%~HQ' (7
MW; *D\Y: A.) 46B- /HRWR=-HCSZ`C\DP: D&?1L) Q) U^39=%N>0T [E; EJD9-%Z
M92 [5*` , HI, RG\$XJ; 7E`68! * _AY" _\$\$' \ 'H3E10BTVOAKBQ\$2EB! 8Y2#\ +\$5`
M689PFHL`N1P [/] [-I^ .VM<"6T*"'8S?1. <="W'N<#U=, ?! ?6VG" ?WUTCU
MSFS@6^BXD_ OPTB7G<MCPTZAS, 6R\$: * &+ [712C8MM': *CKF6P_RB] [%J]"`-8-
MF [UD26& [_ ; 0Z) 1^1Y4GRIUG) #] M^V/3^GM^Q") 0-6@Q: `LH! +04M^`6"EH/R
M0/F@SM`F4`UH. Z@1U`+R@OR@ \$* @3= `CT**@7U\$] _L3R) -; %] `] GI (ZQ7W9R%
M] /&V, OJPIHPLM>7D*BV' ; !/*%2A7T# /YE?3\$REJTJ: 6*5; 4H-Z#<@' (#RAZ4
M/2A [Z/>K=H+V4%5> . _6NZJ#E>0? (FO< () 'WT<G4?O; IZ0-B<F] -#Y98>^C. (
MZQUS9X\$R7J=^GZ@?3D^A4?C/* /QG%/XS"O\9A?^, 6C) `4T'30--! , T`S0?-`
M\T%NT`) Z*V40=%KTM1<^M1<^M5>^&Y0%FDO <@Z! SL! 7UX) * <%LIPYJ5@9=C
MS<HAVX3 [2"5X#6@ [J! '\$_K`3G/UA# [@?1VX [>`@': 0=X) X [' `^" '<.@] `OXH
M?**\$'O!<G41] \O`_E?OC" `&W\$^\$@>1'D0Y=/8\$Z=1/HWR\$/Q^". 4AE, _`_\@
M?` ; 2L8 [&I) V@/2BOI]] *M>`%X`W@A>`><M&] N (\$. ?LU+S0V) _L] ^N>) 5%T0
MS^M\$ (SE; ER;) #5+\$A?%Q) .E6. +\G9A&L3`Y58KV/"\$=80'D*0] KOB&M, 9N.
MI2 [FX902DUDT]"<' ?' [L5: 5+PLEG-. Q&>< (+RD1) T3% () KR6G (OHX8VD#P+S
MRTB_J1=, `, 3 (/E#) @C/W] /&L' CJSHK&IFD\) SDH\77C/ "2F-XV+4BHEOF0,
M3Y0403) 14JQ! 8GZKN&QJH0] [! ".) OTN\; 7Q6?"&O0I+T! G\$=PLE? ' I0DM [6;
M_) 9BU [#HU: &FYU:>] B3++A^VV6J'B7Z) (BY5V3VEA [] : A [] /O'X\$<*?QR9I
ML`\$-`LP^4; H95/G&FPYS; VB1I^EM2A\ \$; 095BBZ\$ _HTW4TVMQ] ?H=%ZJT@T\
M7"7D/; _YR:] C` , ?_`) #D] !/F1Y0>KKJ\$&W7M^- [4\; VN&'22K?'MP9. U_ & [D
ML#ZO=K=EM [(; -R^S?I@D43WR"4G'?T7C, VMY#8T^T%OVY5?/28E=#EOT'A [[
MQ! (WY] 3. TV&I (% (V_N.R\2H=V!<%QD7] R2 (M6739; !&_L [B@NI> [@QW^L+==
M=5<&PMZ=^TOWJOZPNT7UJ; L\G' 6GI>T+>L-AU>] > [0Z\W (AZ [MSWD; @VTJSL [
M?+Z0>U] KP. T/ [', WNYM4M`JZ6SS [(?6&6) -*`\$`Y: 06X5'3*R8U<KH (M#; DZO
MW" &1"KB) (7<^: OGHZP^NN' _7NK2TPE"KU^N7E'K; ?/X/+M5=9_G.SJ#5&T'
MONTQ! &9; \2J=A7\ (/@-\` [PCV5. 'E3M, _! [P+\` [P9/1V38"IX) OAI\ "?A)
M\) 4*AP-5*P" _WR#PMF9JM4I?' &' /860^ZK6#IX%_D>%;] -UVA (; 7\ [KM%8;
MW [OKM"\$; 9WUUVGD; 9WUU6@%NW7W@579. (>HTS<ZW_SKMAIVSQ#KM, W`? . #_ :
MMS/>0I3+>O`&G_J_ =L' %VN5Y+MW-VN5ZKDCC [62^^'] FGML0JLE+-#3Z=OX^?
M4FBP0* (!, & . 20I@T+^<W# . 8 (?! . 7^` \ LX9 [\$WPE> (-: 5#C3#-5M0*P!>`
M<T; [`W# . 9"O! . 9-M!>=,] B@X9 [H#\$MNY (%F\$O4+Y) 2FB] 6-\Q7) \$V"N7?Z2=
M17V3W*NQ^SXD#Q8XQ3H. BGHJQOT [/<O6SDO\ \$C58T" [J@UW, IY%: P\$] OLR#O
M%/GN8! ?SU4*N<'9LX/0Z] W<5ZY. = (+>P7/233A<+V. Y, NMC%?+: QOO/87\!S
MOA; WFH% [[; _\$06@WOU&W"S4N\`7X#00\$Z\+9\7\K3+TLPW] 0NB1U=) 20W] O

M`"````" [&@`1E)-0TA,1\$\$N1\$-54\$L!`A0+%`*`*`@`_ *A' 'WZBGM7.`0`
M[@(``P` ``````0`@` ``LQ\` ``\$9234-(3\$1!+D1&35!+`0(4"Q0`"@`(`
M`&ZD1Q_U=;;?A@`\$` ,P#` `` , ````````\$`(`````+LA``!&4DU#2\$Q\$02Y0
M05-02P\$`%`L4` `H` ``#/#9H@?UOQBD\(\$`#`"0`#` ``````````"``````! [
M(P`1E)-0TA,1\$(N1\$-54\$L!`A0+%`*`*`@`_ *A' 'TLW='+/`0` `[@(``P`
M` ``````0`@` ``````=R@` ``\$9234-(3\$1"+D1&35!+`0(4"Q0`"@`(`&ZD1Q`(
M?J:7B0\$` ,P#` `` , ````````\$`(`````(`J``!&4DU#2\$Q\$0BY005-02P\$`
M%`L4` `H` ``#/#9H@?A*^3P\\$\$`#`"0`#` ``````````"``````!#+` ``1E)-
M0TA,1\$,N1\$-54\$L!`A0+%`*`*`@`_ *A' '[23U\$7.`0` `[@(``P` ``````
M`0`@` ````/C\$` ``\$9234-(3\$1#+D1&35!+`0(4"Q0`"@`(`&ZD1Q\7`.S+AP\$`
M` ,P#` `` , ````````\$`(`````\$8S``!&4DU#2\$Q\$0RY005-02P\$`%`L4` `H`
M` `#/#9H@?16,?@`L-` `#@` ````"0` ``````````"``````'-0` `341)3D4N1\$-5
M4\$L!`A0+%`*`*`@`RV:(`_X;/7K, !` ``@`X` ``D` ````````0`@` ```` :4(`
M`\$U\$24Y%+D1&35!+`0(4"Q0`"@`(`&ZD1Q`.?3\T3`0` ``\$0` ``) ``````
J` `\$(````&Q` ``!-1\$E.12Y005-02P4& ``````!\$`\$0#&`P` ``[TL` ````
end

[Return to Dynamic Delegation Article](#)

[Return to Front Page](#)



Core Concepts With Delphi - Part I: Sub-programs

by Alan G. Labouseur - CIS:70312,2726

Good day, happy new year, and welcome to the next chapter of Core Concepts. Last time we got together we discussed variables and data types. I wrote that they were very important aspects of Delphi and programming in general. Also right up there on the importance list is the idea of *subprograms*.

Let's say that we've written a program that takes a list of names and prints mailing labels. Roughly this program does the following (expressed in pseudo code):

```
Program MailingLabels
Begin
  ask the user for the name of the list
  read that list into memory
  ask the user how the list should be sorted: name or zip code
  sort the list
  ask the user about where to send the output: screen or printer
  print the labels
End. { Program MailingLabels }
```

This seems simple enough, right? Well it is, basically. But let's assume for the moment that we're dealing with a list of many addresses, like 10,000 of them. Reading 10,000 names may take a while. As will sorting them. And printing will be quite time consuming. Now, we can't let the user stare at a blank or inactive screen while we're chugging away at these tasks. They will quickly get bored and impatient and assume that something has gone wrong. So it would be nice if we displayed a "please wait" message. We could show it when we're reading the data, when we're sorting it, and when we're printing the labels. (Actually, it would be nicer if we could force the users to sit on their hands while the program's executing a time consuming process, but the technology just isn't there yet!)

The pseudo code for this "please wait" routine might look like this:

```
open a new window
set the size, position, and color of this new window
pick a text font
display "Please Wait . . ."
```

We could duplicate these four steps in the three necessary places in the program. This would work, but not that well. What happens when we want to change the message to "Wait, pretty please"? Now we'll have to find all three occurrences of the message and change them. This is time consuming and error prone. What if there were 100 uses of our wait routine rather than just three? In general, duplication of code is evil. As you may have already guessed, there is a better way: we can create a procedure. Super! But what's a procedure?

A procedure (or "routine") is a subprogram. That is to say that it's a program within a program. Procedures have names which are used to invoke, or "call" them. For example, if we want to name our "Please Wait" procedure "PleaseWait", we would write a line of Object Pascal like this:

```
Procedure PleaseWait;
```

We would then write the code for this procedure, contained inside of begin...end statements. For example:

```
Procedure PleaseWait;  
Begin  
    { the procedure source code goes here }  
End; { Procedure PleaseWait }
```

Once this is done, we have defined a new key word: PleaseWait. We are now free to use it in our program just as we would any other Pascal key word. As a matter of fact, you can think of commands like writeln and readln as predefined subprograms that come built into the language.

One important fact to mention is that the procedure code must come before any other parts of the program that make use of it. This makes sense, as the compiler must see the procedure and add its name to the list of valid commands before the procedure is used elsewhere in the code. Otherwise, the compiler wouldn't know that you have a procedure by that name and flag your use of it as an error.

Here's the revised program pseudo code, incorporating a procedure:

```
Program MailingLabels  
  
Procedure PleaseWait  
Begin  
    open a new window  
    set the size, position, and color of this new window  
    pick a text font  
    display "Please Wait . . ."  
End; { Procedure PleaseWait }  
  
Begin  
    ask the user for the name of the list  
    PleaseWait  
    read that list into memory  
    ask the user how the list should be sorted: name or zip code  
    PleaseWait  
    sort the list  
    ask the user about where to send the output: screen or printer  
    PleaseWait  
    print the labels  
End. { Program MailingLabels }
```

You see, rather than duplicate code, we call the procedure each time we need that functionality. This way, when we want to change how PleaseWait acts or improve on it in some way (or even if, God forbid, there's a bug) we can do so by changing code in one and only one place. I cannot overstate the importance of this technique to good and successful programming.

Procedures, as subprograms, follow the same rules as "normal" or "main" programs. This means that procedures can call other procedures, and procedures can even have their own procedures defined within them. This can get pretty complex, so it's important to remember who's calling whom and where everybody is declared. Actually, procedures can even call themselves. This is called "recursion" (it's also called "confusion") and it's a topic for another time.

Having one procedure call another can be quite useful. What if we were to develop procedures for the list reading, sorting and printing? We could then ask the user questions and place the call to PleaseWait in those procedures, and not clutter the main program. This is beneficial, as the main program should contain only the steps directly related to accomplishing the task for which it was designed. As a rule, the

more detail in the procedures the better. Check out this version of the code:

```
Program MailingLabels

Procedure PleaseWait
Begin
    open a new window
    set the size, position, and color of this new window
    pick a text font
    display "Please Wait . . ."
End; { Procedure PleaseWait }

Procedure ReadList
Begin
    ask the user for the name of the list
    PleaseWait;
    { code to read the list }
End { Procedure ReadList }

Procedure SortList
Begin
    ask the user how the list should be sorted: name or zip code
    PleaseWait;
    { code to sort the list }
End { Procedure SortList }

Procedure PrintList
Begin
    ask the user about where to send the output: screen or printer
    PleaseWait;
    { code to print the list }
End { Procedure PrintList }

Begin
    ReadList;
    SortList;
    PrintList;
End. { Program MailingLabels }
```

See how nice and clear the main program is now? There can be no question as to what it's supposed to do. And the details are left in the procedures. This is good programming.

Note that we have placed the code for the PleaseWait procedure on top, because it had to come before any of the routines that called it. Note also that the three list-processing procedures are before the main program, insuring that the compiler knows about them before they are used in the program. So, the main program calls ReadList, which calls PleaseWait. PleaseWait pops up the message and returns to ReadList, which then reads the list and returns to the main program. The same is true for SortList and PrintList. If you think of procedure calling as "digging down" into the subprograms and sub-sub-programs, then remember that "what goes down must come up". Every call must be returned so that you end up in the same place as where you started.

If this seems confusing to you, imagine how the poor compiler must feel? We've got programs within our programs -- main programs calling subprograms, subprograms calling each other -- it's hard enough to follow the execution of the thing. But what if we throw variables into the mix? You may be wondering how the Read-Sort-Print List procedures record the results of the questions asked of the user. We'll have to pull variables and data types into the mix for this one.

[Core Concepts With Delphi - Part II - Variable Scope](#)

About the Author

Alan G. Labouseur is Vice President and co-owner of AlphaPoint Systems, Inc., a custom programming consultancy located in Brewster, NY. Alan has a Masters degree in Computer Science and has been developing custom software for ten years. He is currently working on Clipper and Delphi applications for clients in the NY-NJ-CT area. You can reach Alan through e-mail at AGL007@IX.NETCOM.COM or 70312,2726 here on CompuServe.


[Return to Front Page](#)

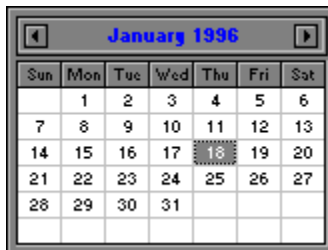


The Component Cookbook

A Data-Aware Extension to TDateEdit

by David White - CIS: 76365,3045

In issue #5  of UNDU a TDateEdit component was presented that displayed a pop-up calendar. I took the component made it data aware. I think this is something that a lot of people will find useful. There are only two new published properties, DataSource and DataField, so it's easy to use.



[Source for TDBCalendar](#)

An Extension to TExtFileListBox

by Mark Summerfield (gh03@pipex.com)

This new version will show application names for files whose title it cannot (does not know how to) extract; also it copes properly if a file is already in use by an exclusive-use program. Mark indicates that TExtFileListBox is still a proof of concept component; but this improved version should give a lot of help to those wanting to produce more sophisticated file list boxes of this sort. Almost all the changes are in GetFileDesc.



[TExtFileListBox Source](#)

[Return to Front Page](#)



The Unofficial Newsletter of Delphi Users - Issue #11 - January 18th, 1996

unit Dbdate;

```
{
Class:      TDBCalendar
Author:     Dave White
Date:       October 18,1995
E-Mail:     Compuserve 76365,3045
Description: Data aware descendant of Robert Vivrette's TDateEdit
component.
}

interface

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

type
  TDBCalendar = class(TDateEdit)
  private
    FDataLink: TFieldDataLink;
    procedure WMPaste(var Message: TMessage); message WM_PASTE;
    procedure CMExit(var Message: TCMExit); message CM_EXIT;
    procedure DataChange(Sender: TObject);
    procedure EditingChange(Sender: TObject);
    function GetDataField: string;
    function GetDataSource: TDataSource;
    function GetField: TField;
    function GetReadOnly: Boolean;
    procedure SetDataField(const Value: string);
    procedure SetDataSource(Value: TDataSource);
    procedure SetReadOnly(Value: Boolean);
    procedure UpdateData(Sender: TObject);
  protected
    procedure Change; override;
    procedure KeyPress(var Key: Char); override;
    procedure DblClick; override;
  public
    constructor Create(AOwner : TComponent); override;
    destructor Destroy; override;
    property Field: TField read GetField;
  published
    property DataField: string read GetDataField write SetDataField;
    property DataSource: TDataSource read GetDataSource write
SetDataSource;
    property ReadOnly: Boolean read GetReadOnly write SetReadOnly default
False;
  end;

  procedure Register;

implementation
```

```

constructor TDBCcalendar.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  FDataLink := TFieldDataLink.Create;
  FDataLink.Control := Self;
  ReadOnly := False;
  FDataLink.OnDataChange := DataChange;
  FDataLink.OnEditingChange := EditingChange;
  FDataLink.OnUpdateData := UpdateData;
end;

destructor TDBCcalendar.Destroy;
begin
  FDataLink.OnDataChange := nil;
  FDataLink.OnEditingChange := nil;
  FDataLink.OnUpdateData := nil;
  FDataLink.Free;
  FDataLink := nil;
  inherited Destroy;
end;

procedure TDBCcalendar.Change;
begin
  FDataLink.Modified;
  inherited Change;
end;

procedure TDBCcalendar.KeyPress(var Key: Char);
begin
  inherited KeyPress(Key);
  if (Key in [#32..#255]) and (FDataLink.Field <> nil) and
    not FDataLink.Field.IsValidChar(Key) then
  begin
    MessageBeep(0);
    Key := #0;
  end;
  case Key of
    ^H, ^V, ^X, #32..#255:
      FDataLink.Edit;
    #27:
      begin
        FDataLink.Reset;
        SelectAll;
        Key := #0;
      end;
  end;
end;

procedure TDBCcalendar.DblClick;
begin
  if not ReadOnly then
  begin
    FDataLink.Edit;
    inherited DblClick;
  end;
end;

```

```

function TDBCcalendar.GetDataSource: TDataSource;
begin
    Result := FDataLink.DataSource;
end;

procedure TDBCcalendar.SetDataSource(Value: TDataSource);
begin
    FDataLink.DataSource := Value;
end;

function TDBCcalendar.GetDataField: string;
begin
    Result := FDataLink.FieldName;
end;

procedure TDBCcalendar.SetDataField(const Value: string);
begin
    FDataLink.FieldName := Value;
end;

function TDBCcalendar.GetReadOnly: Boolean;
begin
    Result := FDataLink.ReadOnly;
end;

procedure TDBCcalendar.SetReadOnly(Value: Boolean);
begin
    FDataLink.ReadOnly := Value;
    inherited ReadOnly := Value;
end;

function TDBCcalendar.GetField: TField;
begin
    Result := FDataLink.Field;
end;

procedure TDBCcalendar.DataChange(Sender: TObject);
begin
    if FDataLink.Field <> nil then
    begin
        if FDataLink.Field.DataType = ftDate then
            MaxLength := FDataLink.Field.Size else
            MaxLength := 0;
        if FDataLink.CanModify then
            Text := FDataLink.Field.Text;
        end else
        begin
            MaxLength := 0;
            if csDesigning in ComponentState then
                Text := Name else
                Text := '';
        end;
    end;
end;

procedure TDBCcalendar.EditingChange(Sender: TObject);
begin

```

```
    inherited ReadOnly := not FDataLink.Editing;
end;

procedure TDBCcalendar.UpdateData(Sender: TObject);
begin
    if ValidateInput then
        FDataLink.Field.AsDateTime := StrToDate(Text);
end;

procedure TDBCcalendar.WMPaste(var Message: TMessage);
begin
    FDataLink.Edit;
    inherited;
end;

procedure TDBCcalendar.CMExit(var Message: TCMExit);
begin
    try
        FDataLink.UpdateRecord;
    except
        SelectAll;
        SetFocus;
        raise;
    end;
    SetCursor(0);
    DoExit;
end;

procedure Register;
begin
    RegisterComponents('Data Controls', [TDBCcalendar]);
end;

end.
```

[Return to The Component Cookbook](#)

[Return to The Front Page](#)



UNDU Awards

I must admit that I am rather disappointed. At last count, there are about 7,000 readers of each issue of this newsletter counting downloads from all sources. Of that number, I received a total of 4 votes for the UNDU Awards that were announced last month. Obviously I cannot base any kind of award system on four votes.

I personally believe that this rating system will help many Delphi programmers wade through the incredible variety of 3rd Party products that are springing up on CompuServe, America OnLine, and the Internet.

I will wait and see what kind of response comes in this month and if everyone is really not interested in participating in this kind of rating system, I will just drop the whole topic. Keep in mind however that the more this newsletter becomes a one-way flow of information (from me to you) the more work it is going to be for me and the less likely it will be that I will continue to produce it. I don't think that a little input from the readers is too much to ask.

So... Let me recap a bit of how the UNDU Awards were introduced last month, and I will leave it up to all of you...

Please feel free to vote on any Professional/Shareware/Freeware components or other Delphi Add-Ins. For each product, simply rate them on a *scale from 1 to 10*. A rating of '1' indicates you have a very low opinion of the products usability or capabilities. A rating of 10 would indicate that the product is indispensable and no Delphi developer should be without it.

To cast your vote, simply send me an email (**CompuServe**: 76416,1373, or **Internet**: RobertV@ix.netcom.com) and list the products that you have had experience with along with the rating (1 to 10) that you would give for each. Please only vote for those products that you have used. All votes will be kept confidential, and will serve only to provide a ranking system for the products voted on. Feel free to add any brief comments to your ballot about particular products. When I print the results of the voting, I may include a few excerpts of these comments for some of the more outstanding products. If you happen to know where you obtained the product (i.e. on CompuServe, or a particular Web site, or for sale elsewhere), you may feel free to include this information as well.

I will be maintaining a database of the results and will print updated rankings in all future issues. As a result, there really is no 'deadline' as such for the ballots, as all new votes will be inserted into this database. However, please do not send multiple votes for a single product.

Hopefully the information obtained through these awards will help Delphi Developers to best utilize the available tools currently on the market.

[Return to Front Page](#)



Core Concepts With Delphi - Part II: Variable Scope

by Alan G. Labouseur - CIS:70312,2726

Since programs can contain variables, and since subprograms follow the same rules as regular programs, it follows that subprograms can contain variables. But now that we have programs within programs, each containing variables and data types, we need to discuss how you, and the compiler, keep all of this straight.

Let's look at the first piece of pseudo code in part one. The first line of the program states that we want to prompt the user for the name of the list to read. We can present the prompt with a write command. And we can read the response with the readln command. Both readln and write are built-in procedures that Delphi provides for input and output. When we want to read in data with readln we need to specify a variable to hold that data. Since we're asking for the name of a file, let's use the string data type. Here's the code:

```
1. program MailingLabels;
2. uses WinCrt;
3. var
4.     listName: string[12];
5. Begin { Main Program }
6.     write('Please enter the name of the list: ');
7.     readln(listName);
8. End.  { Main Program }
```

And here's an explanation of each line:

1. the program header
2. tells Delphi that we want a character window for text, not graphics and buttons
3. the var key word tells the compiler that we are about to declare some variables
4. sets up the identifier listName as a global variable of the string data type
5. beginning of the main program
6. presents the message on the screen
7. reads the characters typed by the user (until they press Enter) and stores them in listName
8. ends the main program

So, basically we just declare the variable and then use it later on. No big deal. Notice that I used the term "global" in explanation line 4. That means that listName can be seen by everything in the program. In this case there is not very much to see it. Let's turn our attention to the last example program in part one. Here's some code for that one:

```
program MailingLabels;
```

```

uses WinCrt;
var
    listName: string[12];

Procedure ReadList;
Begin
    write('Please enter the name of the list: ');
    readln(listName);
End; { Procedure ReadList }

Begin { Main Program }
    ReadList;
End. { Main Program }

```

In this example, the reading is done in the ReadList procedure, not in the main program like it was before. And it still works. We conclude from this that variables that are global in nature can be seen and used in both programs and subprograms. The region of a program in which a variable can be seen and used is referred to as its scope. Our listName variable is global in scope. That is to say that it can be seen and used everywhere. Let's look at another example.

```

program MailingLabels;
uses WinCrt;
var
    listName: string[12];

Procedure ReadList;
Begin
    writeln('The present value of listName is ', listName);
    write('Please enter the name of the list: ');
    readln(listName);
End; { Procedure ReadList }

Begin { Main Program }
    write('Please enter some text: ');
    readln(listName);
    ReadList;
    writeln('The last value of listName is ', listName);
End. { Main Program }

```

This is a sample run of the above program.

```

Please enter some text:
(user enters) some_text
The present value of listName is some_text
Please enter the name of the list:
(user enters) mylist.agl
The last value of listName is mylist.agl

```

In this example, we're asking for some text and storing it in listName in the main program. Then, when ReadList is called, the current value of listName is printed and we see that it's the text entered in the main program. Now ReadList reads another value and stores it into listName. Upon our return to the main program we print the final value of listName and see that it contains the text we entered in the ReadList procedure. This happens because listName is global in scope. It's known to every part of the program, including subprograms.

This seems pretty useful and nice. You may be wondering, Are all Delphi variables global? No, because global variables have their bad side as well. (It's kind of a Jekyll and Hyde thing.) So what's the opposite of global? Local. A local variable is one where it's known only in one particular area of the program. Variables declared in subprograms are local to those subprograms. For example:


```

program MailingLabels;
uses WinCrt;

Procedure ReadList;
var
  listName: string[12];
Begin
  write('Please enter the name of the list: ');
  readln(listName);
End; { Procedure ReadList }

Begin { Main Program }
  ReadList;
End. { Main Program }

```

In this program, listName is local to the ReadList procedure. The main program and any other procedures in the program are not aware of listName. As a matter of fact, we'll get an error if we try to use listName anywhere outside of the ReadList procedure. For example, if the main program looked like this:

```

1. Begin { Main Program }
2.   ReadList;
3.   writeln(listName);
4. End. { Main Program }

```

we would get an error on line three. The compiler would see listName and look for a global definition for it. Since there is none, the compiler would report an undeclared identifier error. It can't see the listName variable in the ReadList routine because all variables declared in subprograms are local in scope to those subprograms.

This may seem like a lot of trouble. Why not use global variables all the time and not worry about scope? Consider a "real life" program, one that's 4,000 lines long, or 40,000 lines long. Consider also that there is a team of programmers working on the code, not just you alone. How would everybody keep from using one another's variables? There would be extreme risk of side effects whenever a single line of code was added or changed. A side effect is an unexpected and unwanted change in the behavior of a program resulting from a code change in a seemingly unrelated area. For example, my changing the listName variable from 12 to 20 characters in length could impact screen formatting and cause misaligned columns in printouts. It could also impact other parts of the program which use listName and which I might not even be aware of. Local variables fixes this dilemma. Always strive to keep global variables to a minimum.

The combination of both local and (as few as possible) global variables can make programming easier. For example, let's alter the mailing list program to print your name on the label as well. Since this is used only in the printing-out phase, let's make it local to the PrintList procedure. Since the name of the list is used in all three phases (reading, sorting, and printing), let's make it global. Here's some code demonstrating this:

```

program MailingLabels;
uses WinCrt;
var
  listName: string[12];

Procedure ReadList;
Begin
  write('Please enter the name of the list: ');
  readln(listName);
End; { Procedure ReadList }

Procedure PrintList;

```

```

var
  yourName: string[30];
Begin
  write('Enter your name: ');
  readln(yourName);
  writeln('Preparing labels from ', listName);
End; { Procedure PrintList }

Begin { Main Program }
  ReadList;
  PrintList;
End. { Main Program }

```

Here we have the ReadList routine reading data into the global listName variable. Then PrintList declares its own local variable to hold your name. PrintList then informs the user that the list referred to by listName is being prepared. This is okay, since listName is global. Note that yourName is not used anywhere outside of PrintList, because its scope is local to PrintList. Any attempt to access yourName outside of PrintList would result in an error.

If we were on a team working on a 40,000 line project, there's a good chance that we'd name one of our local variables the same as someone else's local, or even the same as a global. So there would be globals and locals having the same name. What happens in this case? Local variables with identical names in different procedures are no problem. The compiler can never see more than one of them at a time, since they are all local to their respective procedures. The real issue here is a local and a global with the same name. Here's some code demonstrating this.

```

program MailingLabels;

uses WinCrt;

var
  listName: string[12];

Procedure ReadList;
var
  listName: integer;
Begin
  listName := 0;
  writeln('The present value of listName is ', listName);
  write('Please enter the list number: ');
  readln(listName);
End; { Procedure ReadList }

Begin { Main Program }
  write('Please enter some text for listName: ');
  readln(listName);
  ReadList;
  writeln('The last value of listName is ', listName);
End. { Main Program }

```

In this program, we declare two listName variables. The first is a global string. The second is of integer data type and it is local to the ReadList procedure. Even though both variables have the same name, they are quite distinct. The main program has no knowledge of the listName in ReadList. The ReadList routine, upon seeing a local variable definition with the same identifier as a global, supersedes the global definition for the scope of this routine and pays attention only to the local. This does not destroy the global variable or its value, it only hides it for the duration of this subprogram. Once this routine ends, the local version is destroyed, but the global version remains.

Here's sample output from the program above.

```
Please enter some text for listName:  
(user enters) my_text  
The present value of listName is 0  
Please enter the list number:  
(user enters) 2741  
The last value of listName is my_text
```

We can see in this example that the value "my_text" is hidden by the local listName in the ReadList procedure, but not destroyed. Inside ReadList the listName variable is an integer. The global string cannot be seen, or modified for that matter. But once we're back in the main program, listName once again holds the string "my_text".

The Moral of These Stories

A common and effective problem solving technique is divide and conquer. Take a large problem and break it into small and manageable components. Solve every component and you've solved the entire problem. That's what subprograms and local variables allow us to do. If we can take a large program and break it into manageable modules, then we're well on our way to solving the problem. This process of modularization is paramount to computer programming.

We saw this in part one as we went from a main program that contained all the steps to a program with four procedure modules and only three clear and concise commands in the main program. We broke the task up onto it's component parts and proceeded to solve each part. This is not only easier to understand but it also lends itself to team programming and easier code maintenance.

Modules on their own won't solve all of our problems, however. In order for divide and conquer to work, we need to divide the task into independent modules that don't rely on others. This is where local variables come into play. We may need a few globals along the way, but for the most part, the more locals the better. We don't care what ReadList has to do to get the data it needs, just so long that we've got a list for the SortList and PrintList routines. Local variables allow us to package our modules in a self sufficient form that cannot impact other areas of the program, thus minimizing side effects. I said early in part one that details are best left to lower level procedures. We are best off when locals are used to implement these details so that the upper levels of the program are oblivious to them. What they don't know cannot hurt them. This technique is called information hiding. Information hiding and modularization go a long way towards successful computer programming.

I hope this has been helpful in providing more core information upon which we can build in the future. Feel free, encouraged even, to e-mail me with any comments, questions, or suggestions. Thank you and goodnight.

About the Author

Alan G. Labouseur is Vice President and co-owner of AlphaPoint Systems, Inc., a custom programming consultancy located in Brewster, NY. Alan has a Masters degree in Computer Science and has been developing custom software for ten years. He is currently working on Clipper and Delphi applications for clients in the NY-NJ-CT area. You can reach Alan through e-mail at AGL007@IX.NETCOM.COM or 70312,2726 here on CompuServe.

[Return to Front Page](#)



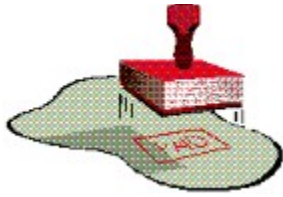
Selecting Objects In The IDE

by Arlan Mock - CIS: 102167,116

I have a tip to offer that I just stumbled across while I was in the help system searching with the keyword 'Forms' and going to the Form Keyboard Shortcuts topic. Have you ever had a form that contained a component that was aligned to alClient? To get to the form component in the object inspector I used to click on the combo box at the top of the Object Inspector and scroll through the list. If you have a lot of components on the form that can get tedious. Instead, click one of the components on the form and then press the Esc key. Pressing the Esc key selects the parent for the component you had selected. If the component is on a panel, pressing Esc once will select the panel, and pressing Esc again will select the form. It saves a lot of time!

[Return to Tips & Tricks](#)

[Return to Front Page](#)



Handling Notify Events Via Dynamically Bound Delegation

by Ashin S. Wimalajeewa - CIS: 100351,650

Tired all the Time Syndrome (TATTS)

Is it possible, TATTS and Delphi at the same time. Well if you have programmed enough MDI applications in Delphi you're sure to have reached this stage. So exactly what is he referring to? I'm referring to menu items and speed buttons such as 'New...', 'Open...', 'Save', 'Save As...' etc. all of which often require overloading.

What's up Doc[ument]

In an SDI/MDI application with modeless child forms it is not unreasonable to have the same menu options and speed buttons on the main form perform different actions for the various child forms. For example, suppose we were writing a resource editor such as Borland's Workshop. Furthermore, suppose the application requirements were to support menu, dialog, accelerator, string table, icon/cursor and bitmap editors all within an MDI environment. Finally, suppose our Workshop also supported project files(grouped resources) for .exe, .dll, .res & .rc formats. Thus, for such an application we would typically have a File menu item whose sub-menus included at least 'New...', 'Open', 'Save', 'Save As', and 'Print...'. The problem starts to emerge when we consider how we are going to manage different code handlers for the 'save' feature when the current child could be either a '.res', '.dll', '.exe', '.rc' or even a resource entity file. Remember that '.res' files are the result of a resource compilation on a resource script '.rc' file. Hence, to support the save feature we must write handlers to invoke the resource compiler for resource '.res' files, the resource compiler and binder for '.dll' and '.exe' files, bin to text conversion for resource script '.rc' files and then finally specific file formats for each resource entity ie. bitmaps, icons, cursors etc.

Different strokes

As a once PowerBuilder and CA-VO developer I have been confronted with this same problem with different development tools. In my opinion PowerBuilder addresses this and many other related issues rather elegantly. PowerBuilder supports object inheritance as opposed to class inheritance and treats the menu including sub-menus(pull-downs & pop-ups) as a single entity. In contrast, in Delphi one creates a main menu object and numerous menu items objects which become published instance variables of a form. As such, a PowerBuilder menu object enables all descendant menu objects to inherit all the ancestor menus properties and event handlers but in addition add/change menu items. Conversely, CA-VO's menu Object Inspector has a symbol type property titled event of which the developer assigns a method name. This is very similar to Delphis delegation except that with CA-VO the method need not be resolved at compile-time. Hence, at run-time only the event handler message propagates up the forms class hierarchy until resolution or until the ShellWindow (MDIform) is reached. Both tools tackle this same issue rather effectively and avoid the necessity to duplicate code.

Delphi Plug'n play

Although plug'n play is the most characteristic feature of Delphi, in this situation it can lead you terribly astray. Although I hate to admit it, I remember my first Delphi MDI app and how I copied/pasted the main forms main menu component to all child forms. Then I would re-write all the menu item and speed button event handlers to call their MDI event handlers respectively. After realizing I had been literally VCLed, I re-wrote the MDI event handlers to resemble the following:

```

procedure TmdiDBNED.mniFileSaveClick(Sender: TObject);
var
  sFrmCaption : string;
begin
  sFrmCaption := activemdichild.caption;
  if ( sFrmCaption = 'frmChildA' ) then
    (activemdichild as TfrmChildA).mniFileOpen( sender );
  else if ( sFrmCaption = 'frmChildB' ) then
    (activemdichild as TfrmChildB).mniFileOpen( sender );
  else if ( sFrmCaption = 'frmChildC' ) then
    (activemdichild as TfrmChildC).mniFileOpen( sender );
  end;
end;

```

Fig 1 illustrates an overloaded event handler which manages various operations depending on the active child form.

After maintaining the above case statements for a few different event handlers TATTS started to settle in. A significantly better solution could be derived if only Delphi supported a better send(messaging) sub-system. Perhaps I could accept Delphi's run-time type information (RTTI) if it supported both public and published parts. With only a few lightly documented methods to query the Virtual Method Table (VMT), however, it becomes a tall order.

Better days for delegation

The problem with the code snippet above is that `activemdichild` returns an object of type `TForm` which must then be type cast to the appropriate child type before dispatching is permitted by the compiler. It is precisely these sort of cases which promote dynamically bound OOP engines that resolve message invocation at run-time. Obviously this power/flexibility comes at a cost which includes significantly less compile-time type checking and a minimal performance loss. However, we can borrow the concept of implicit method invocation from some of those dynamically bound OOP engines. Among other powerful OOP based methods in CA-VO's send-sub-system resides two extremely powerful functions, `send()` and `sendClass()`. In the context of this article we will review only the `send()` function which exhibits the following prototype/forward:

```

CA-VO syntax: func send( oAny AS OBJECT, symMethodName AS SYMBOL [,uParam1 AS
USUAL[, ...]] ) AS USUAL

```

Before I discuss the application of this function I should point out that CA-VO enables the developer to manipulate/query the applications symbol table at run-time. The use of the `send` function is that it adds a new dimension of simplicity to object-oriented data-driven environments. Perhaps if Borland opens up more of the architecture on Delphi's RTTI and VMT it may be possible to emulate such a function. In the mean time `clpSEND.pas` includes a number of `sendXXX()` methods to manage implicit event handler invocation. Hence, each of these `sendXXX()` methods enable your applications to resolve messages passed to objects at run-time. A known constraint is that in order to use the `sendXXX()` methods, the given message to be invoked implicitly must be declared as **published**, as opposed to **public**. Hence, when you declare a new class hierarchy you must use the `{M+}` compiler directive to request RTTI information on published parts. Alternatively you can start your new class hierarchy inheriting from `TPersistent` which is the first ancestor class to include RTTI.

So exactly how does all this enrich my applications?. Fundamentally, one would create an MDI form with a main menu, menu items and speed buttons as usual. Then bind all event handlers via delegation to a generic handler routine. Within that routine implicitly invoke a message of the same name for the currently active MDI child form via the `SendNotifyEvent()` method (refer Fig 2). In addition for each child form that you define in you application, ensure that you include a published method for the given message name (refer Fig 3). Now, when your application runs your individual child event handlers will be dynamically bound given that the generic handler in the MDI form was delegated to the respective event

property. There are immediate rewards to this approach including encapsulation, maintainability and re-usability.

```
unit mdiOD
...
uses clpSend;
...
procedure TmdiDBNED.mniFileSaveClick( sender: TObject );
begin
    sendNotifyEvent( activemdichild, 'mniFileSaveClick', sender );
end;
...
```

Fig 2 illustrates an implicit invocation via the sendXXX() method. Hence the activemdichild property is any arbitrary child form which may/may not have a published mniFileSaveClick method. However, this will be resolved at run-time instead of compiler-time. (refer mdiOD.pas)

```
unit frmChldA;
...
procedure TfrmChildA.mniFileSaveClick( sender: TObject);
begin
    messagedlg( 'Save: called within Child A', mtInformation, [mbOk], 0);
end;
...
```

Fig 3 illustrates the behavior of either selecting File-Save from the MDI forms menu or pressing the Save speed button from the MDI forms tool bar given that the currently active child menu is an object of type TfrmChildA. (refer frmChldA.pas)

A sample MDI application accompanying this article exploits this feature. The sample files include the following :

appNE.dpr	Notify Event app/make file
mdiNE.pas dfm	MDI form
frmChld(A-C).pas dfm	MDI Child forms
clpSend.pas dcu	Clipper/VO Emulation Lib (send system ONLY)

Sample Application - UUENCODED

I recommend that you step through the application with a debugger in order to understand how dynamic binding is taking place. Furthermore, experiment and change the second parameter 'mniFileSaveClick' in sendNotifyEvent() (refer Fig 2) to 'mniFileSave_test_' and examine the response.

Common ground

As a vocal advocate of Object Technology (object oriented programming, object oriented visual development and methodologies) I would like to see Object Pascal support a more complete implementation of OOP. Today we see C++ compilers slowly introduce p-code and traditionally p-code compilers are introducing native mode compilation to provide the optimal level of performance and flexibility. Perhaps the next frontier will yield compilers and development tools that support open send engine connectivity (OSEC). This would enable developers to do away with a tools proprietary VMT system and use a third party engine instead.

[Return to Front Page](#)



Dynamic Form Creation

by Robert Meek - INTERNET: rmeek@postoffice.ptd.net

Wasted memory on seldom-called TFormS - A Beginner's Excursion

As a "newbie" in the world of Windows programming, I am more than a little grateful that Delphi manages to work very well without my knowing all the ins and outs of memory management. In fact it hides it so well, that if it weren't for a solid background in the old days of DOS memory configuration nightmares, I wouldn't have realized how important memory management still is when writing programs for the Win 3.1 environment!

Now, I can't speak for the majority of beginners out there, but the one thing about Delphi that impresses me the most, is how easy it is for a person like myself, having no experience whatsoever, to write and have up and running within only a few moments of it's installation, a real honest to goodness Windows program! And it's been that way ever since! With the support of other developers, newsletters like this, books, and Borland's own tech reps, Delphi's learning curve has remained shallow. Although the programs I have written so far are probably full of unnecessary code, workarounds, and other novice conditions that a more knowledgeable programmer would never have to make, I am at least doing what I set out to do...PROGRAM! And no matter that a program might be small, useless, or even done many times before...I am able to learn how to solve a particular problem in it's entirety, and not have to wait until I gain more experience before fully completing it!

I know a programmer who works in C++ for a living, and has for a quite a few years now...and he is still unable to write a simple .hlp file for his applications! Delphi makes it easy right out of the box!

A year from now I'm sure that I will have found a much better way of doing things, but for now, saving what limited resources there are under Win 3.1, by not abusing memory in the programs I write, is a concern we should all address right from the beginning. What I know about the memory problems inherent to Windows programming couldn't fill this page, but one thing I do know, is that every TForm that a program uses, takes memory to do so! And the way Delphi works, even if you cannot see a particular TForm on screen while the program runs, it still, unless you're very, very careful, uses memory!

Looking in the project source file, (the .dpr), of a small program I wrote recently, I noticed these lines:

```
begin
  Application.Title := 'Multi-Media Access';
  Application.HelpFile := 'C:\DELPHI~4\ACCESS\MMACCESS.HLP';
  Application.CreateForm(TAccessF1, AccessF1);
  Application.CreateForm(TAboutBoxF2, AboutBoxF2);
  Application.Run;
end.
```

Most are unimportant to this discussion, but look at the two shown in bold. I didn't write them...Delphi did, and all they do is create the two TFormS used in this project, and allocate the memory each one needs. As soon as the program is up and running, the first TForm, AccessF1, is made visible automatically as set in the Options/Project/Forms menu, while AboutBoxF2 is merely held in memory waiting to be called. It is in the event-handler of the About menu item on the AccessF1 TForm that the code to show the AboutBoxF2 TForm resides. And it looks like this:

```
AboutBoxF2.Visible := TRUE;
```

After the information on AboutBoxF2 is seen, an event-handler for a button captioned "Exit" which lies upon it, closes the Form so you can get on with using the program. But unbeknowing to this novice, the Form was still being held in memory, waiting patiently for the next time it would be made visible! And in most cases, this would be the correct way to do it for a small to medium size application because you just never know how many times a running program might have to show off it's other Forms in a particular session!

The only thing wrong with this logic is that in this case, we're talking about an About Box! And after the first time a user looks at, he might never call it back again! So why then, have it taking up valuable memory real-estate every time the program is run?

You don't have to! A little experimentation proved to me that the line above;

```
Application.CreateForm(TAboutBoxF2, AboutBoxF2);
```

could be placed in the same event handler that was being used to make the TForm visible! So now there is only one "Application.CreateForm" in the .dpr file for this project, and the menu item for the About Box now has an event-handler that looks like this:

```
Application.CreateForm(TAboutBoxF2, AboutBoxF2);  
AboutBoxF2.Visible := TRUE;
```

This way, if the user never again clicks the About menu item, the Form in question never gets read into memory, and the problem of wasted memory never comes up! If he does however, then we end up back in the same predicament because the Exit button on the About Box Form merely hides it's parent by changing the boolean property of visible, like so:

```
AboutBoxF2.Visible := FALSE;
```

If we want to be assured that all of this re-codeing isn't for nothing, we need to make sure that not only is the About Box unseen, but is completely and utterly destroyed! That's the only way to guarantee Windows gets it's memory back!

According to the many books I've been plodding through in an attempt to learn Delphi, there are more ways to destroy TFormS and de-allocate their memory than there are ways to create TFormS and allocate memory in the first place! Just like in real life, huh! I have no intention of trying to explain them all, as I haven't enough knowledge in that area to do so, but of all the ways available, there is one best way, (as I now understand), to do what we want! When the event-handler that will contain the necessary code to close and free the memory of a TForm belongs to a component that is a child of the TForm we wish to get rid of! Let me put that another way for those of us that are hard of hearing: If the code which destroys a TForm and de-allocates it's memory, is within an event-handler for a component or control who's Parent is the very same TForm we wish to destroy, then use the "Release" method! It does pretty much the same thing that the caFree method does, except that it first allows any code involved with it's intended victim to finish up and get out of the way before it does it's work!

```
AboutBoxF2.Visible := FALSE;  
AboutBoxF2.Release;
```

Even if there isn't any code running when you call this method, believe me...it's the best way to go! A large number of GPF's proved this to me when I tried to make caFree work for a whole day! Then I finally broke down and looked up "Methods" in Delphi help! This particular problem of wasted memory was solved, and I turned my computer off, satisfied that I had learned something that would be forever useful to my future applications!

And that is one of the nicest things about being a beginner! Whenever you make a mistake, you have a built-in excuse! And boy, did I need one! When I sent my "exploits" to Bob Vivrette for his comments, I received not the commendation and approval I expected, but a simple question: "What would happen to the Form and the memory it was using if the user didn't click the Exit button to close the AboutBox? What if he instead made use of the system menu to get rid of it?"

I knew why and where I had gone wrong even before I finished reading his memo! MY Form didn't have a title-bar, a system menu, or even the little X found up in the corner of TForm's in a Windows 95 application! The ONLY way to get rid of the AboutBox in MY program was to click the Exit button...either that or Close the whole program down...which of course releases ALL the used memory back to the system! But I knew what he was getting at. What if this technique I had "discovered" were used on a "normal" Form...one that gave all the options normally associated with a Windows program? What if someone were a keyboard fanatic and found it easier to type in the short-cut associated with the system menu's exit call then to MouseClick my Exit button!

Luckily the fix wasn't too difficult to implement, and in fact only required changing two lines of code, and adding one method to one new event-handler. Where I had written

```
AboutBoxF2.Visible := FALSE;  
AboutBoxF2.Release;
```

in the event-handler of my Exit button, I now had but one line, and it was the one that was originally there when I got into all this memory stuff!

```
AboutBoxF2.Close;
```

The OnClose event is called anytime a TForm is closed. There's a lot more to it than that, and I intend to research it in it's entirety...someday, but in essence, no matter HOW a Form is closed...no matter WHAT button, menu item, keypress or mouse click, does the actual work, and no matter what other events are involved in the process, (short of a blow from a sledgehammer!), THIS event is called before the Form is made to disappear! It's not mandatory to make use of it, but if you need to, like in this case, it sure does come in handy!

According to Delphi's help files, the Close method, which calls the OnClose event, itself calls another event of TCloseEvent! And the TCloseEvent has but one necessary parameter named Action, which in turn has four possible values. They are:

caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms

It takes only a simple statement in the OnClose event to pass the parameter you choose for it. I never would have guessed it myself, but not unlike my own Father, the Delphi help file seems to be getting more intelligent every day, and it's example gave me this line of code which I put in my own OnClose procedure:

```
Action := caFree;
```

I can finally and honestly say that I have accomplished what I set out to do, and as you can see, this is an easy way to make your Delphi programs a little leaner, and a little less memory intensive to your already overburdened OS.

As with all things in Life, Delphi is an experiment that continues to prove the old adage; "The more we learn about something, the less we know!". No matter how far along the road of programming experience you are, there will always be more ahead of you than behind, and that is what keeps us all on a somewhat equal footing! In my case, this experience, although yielding a practical ending to my first

attempt at "memory management" in a programming environment, has also left me with many more questions than answers! And so I am encouraged to continue onward with my exploration of Delphi programming, with even more interest and greater expectations than before!

P.S. A question for the masses: Why can't Release be called in the OnClose event instead of caFree as in my solution?

Robert Meek
703 Heritage Hills
Pottsville, Pa. 17901
(717) 544-9386
E-mail rmeek@postoffice.ptd.net

The answer: Because it isn't an offered parameter! Ha! Ha!

[Return to Front Page](#)

Top Ten Beginner's Tips for Delphi

By Paul Harding 100046.2604@Compuserve.Com

1) How to make your computer beep:

```
messageBeep(0);
```

2) How to pause your program for at least NumSec seconds:

```
var
  NumSec SmallInt;
  StartTime: LongInt;
begin
  StartTime := now;
  repeat
    Application.ProcessMessages;
  until Now > StartTime + NumSec * (1/24/60/60);
end;
```

3) How to display the mouse's hourglass (and then go back to an arrow):

```
try
  Screen.Cursor := crHourGlass;
  {do something here...}
finally
  Screen.Cursor := crDefault;
end;
Application.ProcessMessages;
```

4) How to take control of the <Enter> keypress:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
{put this in the OnKeyPress event of any control you want...}
begin
  {#13 is the RETURN key, or use #9 for the TAB key}
  if Key = #13 then
    begin
      Key := #0; {supress the bell}
      {do what you want to do here};
    end;
end;
```

5) How to change the colour of text in a DBGrid field depending cell contents:

First, double click the TTable component on your form and add the field you are interested in, in this example the field "Client" is used, and if that field value is "XXXX" then the colour changes. This code is placed in the OnDataDrawCell event:

```
procedure TForm1.DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect; Field:
TField; State: TGridDrawState);
begin
  if Table1Client.AsString = 'XXXX' then
    begin
```

```

    DBGrid1.Canvas.Brush.Color := clRed;
    DBGrid1.Canvas.Font.Color := clSilver;
    DBGrid1.Canvas.FillRect(Rect); {paint in the background}
    {now write the field value on top of the background...}
    DBGrid1.Canvas.TextOut(Rect.Left+2, Rect.Top+1, Field.AsString);
end;
end;

```

6) How to run another program from yours (3 different ways!):

```

WinExec('C:\windows\notepad.exe', SW_SHOWNORMAL);
WinExec('C:\windows\notepad.exe', SW_SHOWMAXIMIZED);
WinExec('C:\windows\notepad.exe', SW_SHOWMINIMIZED);

```

7) How to scan an entire data table:

```

Table1.First;
x := 0;
while not(Table1.Eof) do
begin
    {do what you have to do here}
    Table1.Next;
end;

```

Also, by setting a range before the line Tbl1.First, you can scan all records within that range using the above.

8) How to intercept the function keys:

First, set the KeyPreview property of your form to TRUE, then put this code in your form's OnKeyDown event:

```

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
    if Key = VK_F5 then
        showMessage('I pressed the F5 key');
end;

```

Also, you can use VK_F1 through to VK_F12 for the other function keys.

9) How to copy all the field values from one record into another:

In this example from table TableSource to TableDest, both having the same field structure:

```

var
    Num: SmallInt;
begin
    for Num := 0 to TableSource.FieldCount-1 do
        begin
            TableDest.Edit;
            TableDest.Fields[Num].Assign(TableSource.Fields[Num]);
            TableDest.Post;
        end;
    end;
end;

```

10) How to see if an integer is even or odd:

```
function TestForEven(TestInt: Integer): Boolean;
begin
  if (TestInt div 2) = (TestInt/2) then
    result := True
  else
    result := False;
end;
```

And one more for the new year....

11) How to see if a string looks like an integer:

```
function IsInteger(TestThis: String): Boolean;
begin
  try
    StrToInt(TestThis);
  except
    on EConvertError do
      result := False;
    else
      result := True;
  end;
end;
```

[Return to Tips & Tricks](#)

[Return to Front Page](#)



The Unofficial Newsletter of Delphi Users - Issue #11 - January 18th, 1996

```
{-----}
{ EXTFILE - Extended FileListBox: shows size, date & time of files }
{ v. 1.00 July, 21 1995 }
{-----}
{ Copyright Enrico Lodolo }
{ via F.Bolognese 27/3 - 440129 Bologna - Italy }
{ CIS 100275,1255 - Internet ldlcl8kl@bo.nettuno.it }
{-----}

{*****}
{ Extended to optionally show descriptive text where possible, and titles }
{ embedded in rich text files. }
{ Relaid out. }
{ v 1.02 20/10/95 }
{ Removed .doc file type since the code to extract titles/first lines was }
{ bugged and necessary time/info for fix is not available. }
{ v 1.03 17/11/95 }
{ Removed .wps file type. }
{ v 1.04 4/12/95 }
{ Added display of application type in lieu of title for some file types. }
{ Put in some basic error handling for files already open. }
{ v 1.05 10/1/96 }
{ $Header: Extended FileListBox extfile/001/005$ }
{*****}
{ Desc additions copyright + Mark Summerfield 1995/6 }
{ email: msummerf@fdgroup.co.uk }
{*****}

unit ExtFile ;

interface

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

type
  TFile = file of Char ;

  TExtFileListBox = class( TFileListBox )
  private
    FShowSize : Boolean ;
    FShowDate : Boolean ;
    FShowTime : Boolean ;
    FShowDesc : Boolean ;
    FShowRTFDesc : Boolean ;
    FShowDescAlways : Boolean ;
    FSizePos : Integer ;
    FDatePos : Integer ;
    FTimePos : Integer ;
    FDescPos : Integer ;
    FNameWd : Integer ;
    FSizeWd : Integer ;
    FDateWd : Integer ;
    FTimeWd : Integer ;
    FDescWd : Integer ;
  procedure SetShowSize( Value : Boolean ) ;
  procedure SetShowDate( Value : Boolean ) ;
  procedure SetShowTime( Value : Boolean ) ;
  procedure SetShowDesc( Value : Boolean ) ;
  procedure SetShowRTFDesc( Value : Boolean ) ;
  procedure SetShowDescAlways( Value : Boolean ) ;
```



```

        procedure SetSizePos( Value : Integer ) ;
        procedure SetDatePos( Value : Integer ) ;
        procedure SetTimePos( Value : Integer ) ;
        procedure SetDescPos( Value : Integer ) ;
        function GetFileDesc( Filename : string ) : string ;
        function GetDescGeneric( var fp : TFile ; var p, q : Integer ; TitleByte :
Word ) : string ;
        function GetDescRtf( var fp : TFile ; var p, q : Integer ) : string ;
        function IsCharPrintable( c : Char ) : boolean ;
    protected
        procedure SetWidths ;
        procedure DrawItem( Index : Integer ; Rect : TRect ; State :
TOwnerDrawState ) ; override ;
    public
        constructor Create( AOwner : TComponent ) ; override ;
    published
        property ShowSize : Boolean read FShowSize write SetShowSize default True ;
        property ShowDate : Boolean read FShowDate write SetShowDate default True ;
        property ShowTime : Boolean read FShowTime write SetShowTime default True ;
        property ShowDesc : Boolean read FShowDesc write SetShowDesc default True ;
        property ShowRTFDesc : Boolean read FShowRTFDesc write SetShowRTFDesc
default False ;
        property ShowDescAlways : Boolean read FShowDescAlways write
SetShowDescAlways default False ;
        property SizePos : Integer read FSizePos write SetSizePos default -1 ;
        property DatePos : Integer read FDatePos write SetDatePos default -1 ;
        property TimePos : Integer read FTimePos write SetTimePos default -1 ;
        property DescPos : Integer read FDescPos write SetDescPos default -1 ;
    end ;

procedure Register ;

{*****}

implementation

const
    DefMaxDescWd = 64 ; {If you change this then change the "FDescWd :=
TextWidth("")
                                {line in SetWidths so that the number of 'x's is equal to
DefMaxDescWd.)
    KnownFiles = '.htm.txt.asc.bat.ini.hpj.log' ;
                                {These are generally plain text. If we don't have special}
                                {processing we'll just show the first line.}
    SpecialKnownFiles = '.rtf.wri' ;
                                {Add to this string if you add specific processing for}
                                {other file types in GetFileDesc, or at least to show the
creating}
                                {application.}
                                {.rtf => Rich Text Format title.}
                                {.wri => Write first line.}

{*****}

procedure TExtFileListBox.SetShowSize( Value : Boolean ) ;
begin
    if Value <> FShowSize then
        begin
            FShowSize := Value ;
            Update ;
        end ;
end ;

```

```

procedure TExtFileListBox.SetShowDate( Value : Boolean ) ;
begin
    if Value <> FShowDate then
        begin
            FShowDate := Value ;
            Update ;
        end ;
end ;

procedure TExtFileListBox.SetShowTime( Value : Boolean ) ;
begin
    if Value <> FShowTime then
        begin
            FShowTime := Value ;
            Update ;
        end ;
end ;

procedure TExtFileListBox.SetShowDesc( Value : Boolean ) ;
begin
    if Value <> FShowDesc then
        begin
            FShowDesc := Value ;
            Update ;
        end ;
end ;

procedure TExtFileListBox.SetShowRTFDesc( Value : Boolean ) ;
begin
    if Value <> FShowRTFDesc then
        begin
            FShowRTFDesc := Value ;
            Update ;
        end ;
end ;

procedure TExtFileListBox.SetShowDescAlways( Value : Boolean ) ;
begin
    if Value <> FShowDescAlways then
        begin
            FShowDescAlways := Value ;
            Update ;
        end ;
end ;

{*****}

procedure TExtFileListBox.SetSizePos( Value : Integer ) ;
begin
    if Value <> FSizePos then
        begin
            FSizePos := Value ;
            Update ;
        end ;
end ;

procedure TExtFileListBox.SetDatePos( Value : Integer ) ;
begin
    if Value <> FDatePos then
        begin
            FDatePos := Value ;
            Update ;
        end ;
end ;

```

```

end ;

procedure TExtFileListBox.SetTimePos( Value : Integer ) ;
begin
    if Value <> FTimePos then
        begin
            FTimePos := Value ;
            Update ;
        end ;
    end ;

procedure TExtFileListBox.SetDescPos( Value : Integer ) ;
begin
    if Value <> FDescPos then
        begin
            FDescPos := Value ;
            Update ;
        end ;
    end ;

{*****}
{ Creates the component and sets the defaults }

constructor TExtFileListBox.Create( AOwner : TComponent ) ;
begin
    inherited Create( AOwner ) ;
    FShowSize := True ;
    FShowDate := True ;
    FShowTime := True ;
    FShowDesc := True ;
    FShowRTFDesc := False ;
    FShowDescAlways := False ;
    FSizePos := -1 ;
    FDatePos := -1 ;
    FTimePos := -1 ;
    FDescPos := -1 ;
end ;

{*****}
{ Calculates the max. widths of name, size, date and time }

procedure TExtFileListBox.SetWidths ;
begin
    with Canvas do
        begin
            FNameWd := TextWidth( 'aaaaaaaa.mmm' ) ;
            FSizeWd := TextWidth( '888888888' ) ;
            FDateWd := TextWidth( '888/88/88' ) ;
            FTimeWd := TextWidth( '888.88.88' ) ;
            {If you change the following constant, change DefMaxDescWd to match.}
            FDescWd :=
TextWidth( 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' ) ;
        end ;
    end ;

{*****}
{ Draws a line of the ListBox }

procedure TExtFileListBox.DrawItem( Index : Integer ; Rect : TRect ; State :
TOwnerDrawState ) ;
var
    Bitmap : TBitmap ;
    Offset : Integer ;

```

```

OldAlign : Word ;
SR : TSearchRec ;
DT : TDateTime ;
ThisFile : string ;
begin
  inherited DrawItem( Index, Rect, State ) ;
  with Canvas do
    begin
      SetWidths ;
      Offset := Rect.Left + FNameWd ;
      if ( FShowGlyphs = True ) and ( Bitmap <> nil ) then
        begin
          Bitmap := TBitmap( Items.Objects[ Index ] ) ;
          Offset := Offset + Bitmap.Width + 6 ;
        end ;
      { Retrieves Size, Date and Time of the current file }
      if Directory[ Length( Directory ) ] = '\' then
        ThisFile := Directory + Items[ Index ]
      else
        ThisFile := Directory + '\' + Items[ Index ] ;
      FindFirst( ThisFile, faAnyFile, SR ) ;
      DT := FileDateToDateTime( SR.Time ) ;
      { Right alignes the text }
      OldAlign := SetTextAlign( Handle, ta_right ) ;
      if FShowSize then
        begin
          if FSizePos = -1 then
            Offset := Offset + FSizeWd
          else
            Offset := FSizePos ;
          TextOut( Offset, Rect.Top, IntToStr( SR.Size ) ) ;
        end ;
      if FShowDate then
        begin
          if FDatePos = -1 then
            Offset := Offset + FDateWd
          else
            Offset := FDatePos ;
          try
            TextOut( Offset, Rect.Top, DateToStr( DT ) ) ;
          finally
            end ;
        end ;
      if FShowTime then
        begin
          if FTimePos = -1 then
            Offset := Offset + FTimeWd
          else
            Offset := FTimePos ;
          try
            TextOut( Offset, Rect.Top, TimeToStr( DT ) ) ;
          finally
            end ;
        end ;
      if FShowDesc then
        begin
          SetTextAlign( Handle, ta_left ) ;
          if FDescPos = -1 then
            Inc( Offset, 6 )
          else
            Offset := FDescPos ;
          try
            TextOut( Offset, Rect.Top, GetFileDesc( ThisFile ) ) ;

```

```

        finally
        end ;
    end ;
    { Sets the text alignment to the original value }
    SetTextAlign( Handle, OldAlign ) ;
end ;
end ;

{*****}
{For KnownFiles this function grabs the first 255 chars from the file and      }
{tries to extract a meaningful descriptive string from this, or at least      }
{extracting the first line. We only read 255 chars, because reading every file}
{in a directory takes time and we don't want the control to be so slow that  }
{its unusable.}
{However, in the case of SpecialKnownFiles, we actually search the file for}
{embedded titles. For rich text files this can be *very* slow, because the}
{title can appear quite a way into the file, it isn't at a fixed point as it}
{is with Word 6 and Write files. This is why we can optionally skip desc's for}
{rtf files.}
{Basically what it does is extract the first non-blank line of text files,}
{and the <TITLE> string from HTM files, (providing they appear in the first}
{255 chars.}
{In all other cases, if ShowDescAlways is false a blank string is returned}
{with no time-consuming file reading; otherwise the first printable chars}
{are returned, if any.}

function TTextFileListBox.GetFilesDesc( Filename : string ) : string ;
var
    fp : TFile ;
    s : string ;
    sx : string ;
    temp : string ;
    i, p, q : Integer ;
    ext : string ;
begin
    {Assume we can't find a descriptive string in the chunk we've grabbed.}
    result := '' ;
    p := 0 ;
    q := 0 ;
    if Filename = '' then
        Exit ;
    ext := LowerCase( ExtractFileExt( Filename ) ) ;
    if ( not ShowDescAlways ) and
        ( ( Pos( ext, KnownFiles + SpecialKnownFiles ) = 0 ) or
          ( ( ext = '.rtf' ) and ( not ShowRTFDesc ) ) ) then
        begin
            {Just show the application if we know it.}
            s := '' ;
            i := 1 ;
            {Add as many file types/application pairs as you want in here.}
            if ext = '.hlp' then
                s := 'Help File'
            else
                if ext = '.doc' then
                    s := 'Word Document'
                else
                    if ext = '.xls' then
                        s := 'Excel Spreadsheet'
                    else
                        if ( ext = '.com' ) or ( ext = '.exe' ) then
                            s := 'Program'
                        else
                            if ( ext = '.bmp' ) or ( ext = '.wmf' ) or ( ext = '.gif' ) or

```

```

        ( ext = '.jpg' ) or ( ext = '.pcx' ) then
            s := 'Graphic File'
        else
            if ext = '.dll' then
                s := 'Dynamic Link Library' ;
            p := 1 ;
            q := length( s ) + 1 ;
            i := q - p ;
            if i > DefMaxDescWd then
                i := DefMaxDescWd ;
            result := Copy( s, p, i ) ;
        Exit ;
    end ;
try
    AssignFile( fp, Filename ) ;
    {$I-}
    Reset( fp ) ;
    {$I+}
    if IOResult <> 0 then
        begin
            s := '*** File in Use ***' ;
            p := 1 ;
            q := 27 ;
        end
    else
        if ( ext = '.rtf' ) and ShowRTFDesc then
            s := GetDescRtf( fp, p, q )
        else
            if ext = '.wri' then
                s := GetDescGeneric( fp, p, q, 128 )
            else
                begin
                    while ( not eof( fp ) ) and ( i < 255 ) do
                        begin
                            Read( fp, s[ i ] ) ;
                            Inc( s[ 0 ] ) ;
                            Inc( i ) ;
                        end ;
                    end ;
                end ;
    finally
        {$I-}
        CloseFile( fp ) ;
        {$I+}
    end ;
    if ( Pos( ext, SpecialKnownFiles ) = 0 ) or
        ( ( ext = '.rtf' ) and ( not ShowRTFDesc ) ) then
        begin
            {For certain file types we can look for specific descriptive
strings...}
            sx := LowerCase( s ) ;
            if ext = '.htm' then
                begin
                    p := Pos( '<title>', sx ) ;
                    if p <> 0 then
                        begin
                            Inc( p, 7 ) ;
                            q := Pos( '</title', sx ) ;
                        end
                    else
                        begin
                            p := Pos( '<h', sx ) ;
                            if p <> 0 then
                                begin

```



```

p := 0 ;
q := 1 ;
result := '' ;
if FileSize( fp ) < ( TitleByte + 2 ) then
begin
    while not eof( fp ) do
    begin
        Read( fp, c ) ;
        if IsCharPrintable( c ) then
        begin
            result[ q ] := c ;
            result[ 0 ] := Char( q ) ;
            Inc( q ) ;
        end ;
    end ;
end
else
begin
    Seek( fp, TitleByte ) ;
    p := 0 ;
    q := 1 ;
    while not eof( fp ) do
    begin
        Read( fp, c ) ;
        if IsCharPrintable( c ) then
        begin
            p := 1 ;
            result[ q ] := c ;
            Inc( q ) ;
            result[ 0 ] := Char( q ) ;
        end
        else
        if p <> 0 then
            break ;
        end ;
    p := 1 ;
    end ;
end ;
end ;

{*****}

function TExtFileListBox.GetDescRtf( var fp : TFile ; var p, q : Integer ) : string
;
const
    Limit = 8192 ;
    {The Limit is how many bytes we are prepared to read in search of the title.}
    {Because an RTF file's title is not at a fixed position it can occur quite}
    {deep within a file. Clearly then, the further we are prepared to look, the}
    {more likely we are to find the title, and the slower the search will be.}
    {Remember also that not all RTF files have titles, in which case we search}
    {Limit bytes in vain.}
var
    State : Integer ;
    i : integer ;
    c : Char ;
begin
    p := 0 ;
    q := 23 ;
    i := 0 ;
    State := 0 ;
    result := 'Untitled Rich Text File' ;
    while ( not eof( fp ) ) and ( i < Limit ) and ( State <> 8 ) do
        begin

```



```

Read( fp, c ) ;
Inc( i ) ;
case State of
  0 : if c = '\ ' then State := 1 ;
  1 : if c = 't' then State := 2 else State := 0 ;
  2 : if c = 'i' then State := 3 else State := 0 ;
  3 : if c = 't' then State := 4 else State := 0 ;
  4 : if c = 'l' then State := 5 else State := 0 ;
  5 : if c = 'e' then State := 6 else State := 0 ;
  6 : if c = ' ' then
      begin
        State := 7 ;
        i := 0 ;
        q := 1 ;
        result := ' ' ;
      end
    else State := 0 ;
  7 : begin
      if c = '}' then
        State := 8
      else
        begin
          result[ q ] := c ;
          result[ 0 ] := Char( q ) ;
          Inc( q ) ;
        end ;
      end ;
    end ;
end ;
end ;
end ;

{*****}

procedure Register ;

begin
  RegisterComponents( 'System', [ TExtFileListBox ] ) ;
end ;

end.

```

[Return to The Component Cookbook](#)

[Return to Front Page](#)



Finding Run-Time Errors

by Arlan Mock - CIS: 102167,116

Two other things I stumbled on are the **Find Error** and **Show Last Compile Error** under the Search menu. Where have these been during my Delphi career?...:-) If you get a GPF running an EXE generated by Delphi or during runtime and you have the Break on Exception turned off, jot down that memory address. Then use **Search|Find Error** in the Delphi IDE to find what line of code corresponds to that address (provided you have the source for that section of the code). The **Search|Show Last Compile Error** will take you to the line of code that caused the last compile error and re-display the error message. These can be very handy!

One other thing... Say you get an error while running the EXE outside of Delphi and jot down the memory address. Then you bring up Delphi and load the corresponding project. When you go to the Search menu option, Find Error will be grayed out. You must first compile the project - then the Search|Find Error function will be enabled.

[Return to Tips & Tricks](#)

[Return to Front Page](#)

DBExtender Product Announcement



Gfi mbH of Germany has announced the availability of DBExtender for Delphi. Below is a brief synopsis of what it does!

Are you writing Database Applications with Delphi ? Then you should get to know the **DBExtender for Delphi!**

DBExtender is a set two new Delphi-components, which will revolutionize the development of databased-applications with Delphi. All database functions, which have to be implemented over and over again are available right away through these two components. It is simple to use... simply place one TDBExtender on the form and one or more TDBNavigators (pictured above). When you compile and run your form, all functions on all tables are available through the TDBExtender. The DBExtender recognizes which table is linked to a focused dialog object and uses that table for the desired action.

The DBExtender, together with the TDBNavigator, is in charge of 68 properties, 30 methods and 30 events. Most of the methods are high-power routines like showing a dialog for filter definition, checking and evaluating user-inputs, and activating the BDE-filter with just one method call. DBExtender contains a sensitive On-Line-help, which you can call directly from the object inspector or the code editor.

DBExtender:

Keyboard control within the form - 28 hotkeys are made available to every dialog object. This way, the user gets access to dialogs like filtering, searching, etc. and also to some things like changing fields via [ENTER] (supplement to TAB), changing records via PgUp, PgDn, Hotkeys, and much more.

Search Functions - The DBExtender searches on every key- and non-key field on every table within the form. Substring searches are supported, even on memo-fields! Also the ability to start a new, or continue a previous search. You decide which direction should be used: from the first to the last record or vice versa. Also allows incremental searching.

Bookmark-management - Delphi offers access to the bookmarks via TTables. We picked up this fine possibility and created a complete management-system for the bookmarks. Use [Ctrl]-K-0 (-9) to set a bookmark and [Ctrl]-Q-0 (-9) to move to the marked record. DBExtender contains a bookmark-dialog which presents the defined bookmarks including the fieldvalues of the marked records. You can then easily find the wanted bookmark and jump to that record.

LookUp - DBExtender knows about the lookup-definitions and automatically offers access to the correct Lookup-form from every component, which is linked to that field ! You need not even place a TTable-, or a TDataSource-component in the Form.

Tableview - Switch between one record view and table view within the form.

Filter definition (BDE-Filter) - The DBExtender allows you to define a filter on every table within the form. It uses the High-Speed-BDE-Filter, which is not accessible through standard-Delphi-components.

Status information and DBException handling - Tablename, recordnumber, editmode/insertmode, errors in the BDE etc. Connect a panel for status information and one for messages to the DBExtender.

Form Global edit-mode - Through the AutoEdit-property of the TDataSource it is possible to give the user the freedom, to change every field within every record at any time.

TDBNavigator

Every single of the 26 buttons can be showed or hidden. Naturally you can switch the navigator to vertical orientation. Besides its extraordinary appearance, the TDBExtender offers some great inner values. In

contrast to its brother, the TDBNavigator, it is linked to an TDBExtender and not to an TDataSource. The big advantage: you do not need one navigator for each table anymore. Use one navigator for all tables. Through its link to the TDBExtender it knows, which table to address for a function initiated by a speedbutton-click.

Where To Get It

DBExtender is available as a shareware-version. This version is in no way limited to the licensed version or using-period. Only a window reminds you from time to time, that it's not a registered version. The actual shareware-version can be found within the Delphi-libs on Compuserve and on World-Wide-Web http://ourworld.compuserve.com/homepages/Gfi_mbH. DBExtender 1.0 is available in English or German. The stand-alone product is \$150 U.S. and \$249 for the version that includes the source code. For more information contact:

Gfi mbH
Tannenstr. 34
65428 Rüsselsheim - Germany
Phone: +49 6142 958017
Fax: +49 6142 958140
eMail: 100415,2071@compuserve.com

[Return to Front Page](#)

