maXbox Starter 80

MY C, PASCAL & DELPHI NOTES

Components at Runtime

To add a component to a TabbedNotebook page at run-time a pointer to the desired page must be assigned to the new component's Parent property before it can be shown. The way to access all the pages of a TTabbedNotebook at run-time is with the Objects array property of the TabbedNotebook's Pages property. In other words, the page components are stored as objects attached to the page names in the Pages string list property. The follow demonstrates the creation of a button on the second page of TabbedNotebook1:

```
var
  NewButton : TButton;
begin
  NewButton := TButton.Create(Self);
  NewButton.Parent := TWinControl(TabbedNotebook1.Pages.Objects[1])
  ...
```

This is how a TNotebook page would be used as a parent to a newly created component on that page:

```
NewButton.Parent := TWinControl(Notebook1.Pages.Objects[1])
```

This is how a TTabSet tab page would be used as a parent to a newly created component on that tab page:

```
NewButton.Parent := TWinControl(TabSet1.Tabs.Objects[1])
```

Inherit a property

A: All descendents of TCustomControl have a Canvas property, however, most are protected to prevent 'outsiders' from drawing on the component. Descendents of a component can always access the protected properties they inherit from the component (such as Canvas), but users of the component cannot.

```
type
  TCanvasPanel = class(TPanel)
  public
    property Canvas;
  end;
```

Save an Integer with a string in an Tstring list object

A: Yes, but it requires some type conversions. The TString component has an Objects array along with the string array that can be utilized for the purpose of storing integer datA: The data type that the Objects array holds is TObject. In essence it holds a 4 byte pointer value. So to put an integer value in it you would need to type cast that value. For example, the following is adding a string and an integer value of 100 to an items property (TString object) of a Listbox:

```
Listbox1.Items.AddObject('Text string', TObject(100));
```

To get the value out do the following:

```
Result := LongInt( Listbox1.Items.Objects[0] );
```

This assumes that Result is of type Longint and that the value that were after is at index position 0.

TTabbedNotebook use much of the system resources?

A: Even though only one page is showing at a time each pages' components have already been created thus taking resources. One solution to this is instead of using a notebook you use a separate form for each page and when the user clicks on a tab, the existing page is destroyed and the new one created. The basic steps to set this is is as follows:

First, each child form needs its creation parameters setup in a certain way:

```
...
private
  { Private declarations }
  PROCEDURE CreateParams(VAR Params: TCreateParams); override;
...
procedure TForm2.CreateParams(VAR Params: TCreateParams);
begin
  Inherited  CreateParams(Params);
with Params do begin
    WndParent := Application.MainForm.Handle;
    Style := (Style OR WS_CHILD) AND NOT (WS_POPUP);
  end;
end;
```

The child form's BorderStyle must be set to bsNone. In the main form, create a private data field Child of type TForm. Initialize it in the OnActivate event, NOT OnCreate. And each time the tab is clicked to change "pages", free the existing Child and initialize a new one of the desired type. E.g. in OnActivate do:

```
Child := TForm2.Create(Self);
with Child do begin
  Parent := Self;
  Align := alClient;
  Visible := True;
end;
```

When you create a child page due to a tab change, do it in the same way just shown. Of course you'll need to use the main form to store any data about the state of the controls in a given child window, because the data will go bye bye when the child is freed.

Multitask enabled Delay

```
StartTime: real;

begin
   StartTime:=Time;
   Repeat
     WaitMessage;
   Until Time > StartTime + 1*(1/24/60/60);

procedure TForm1.Delay(msecs:integer);
var
  FirstTickCount:longint;
begin
    FirstTickCount:=GetTickCount;
    repeat
         Application.ProcessMessages; {allowing access to other
                                  controls, etc.}
    until ((GetTickCount-FirstTickCount) >= Longint(msecs));
end;
```

Fast Randomwords to fill Lists

```
for t:= 1 to 50 do
   begin
     newWord:=chr(random(25)+65)+chr(random(25)+65)+chr(random(25)+65)
             +chr(random(25)+65)+chr(random(25)+65)+chr(random(25)+65);

     Listbox1.Items.Add(newWord);
```

Kommentar im Kommentar

```
(*  {}  {}   {}  *)
```

Create a Paradox table with an Auto Increment type field

programatically? I'm using TTable.CreateTable, but TFieldType doesn't include this type."

A: Use a TQuery and SQL CREATE TABLE statement. For example:

```
procedure TBenchDlg.CreateBtnClick(Sender: TObject);
begin
  with Query1 do
   begin
     DatabaseName := 'C:\DATEN\CALWIN2\CWTests';
     with SQL do
       begin
       Clear;
       Add('CREATE TABLE "pBench.db" (ID AUTOINC,');
       Add('FIRSTNAME CHAR(50),');
       Add('LASTNAME CHAR(50),');
       Add('PHONE INTEGER,');
       Add('PRIMARY KEY(ID))');
       ExecSQL;
       Clear;
       Add('CREATE INDEX ByName ON "pBench.db" (LASTNAME)');
       ExecSQL;
     end;
  end;
end;
```

Fill created table at runtime

```
procedure TBenchDlg.FillBtnClick(Sender: TObject);
 var k,err: integer;
 s: string;
begin
 val(Edit1.Text,maxArray,err);
 t:=TTable.create(self);
     with t do
     begin
       DatabaseName := 'C:\DATEN\CALWIN2\CWTests'; {personal alias}
       TableName := 'pbench.db';
       open;
       for k:= 1 to maxarray do
       begin
         str(k,s);
         append;
         FieldByName('Firstname').AsString := 'CWObjekt '+s;
         FieldByName('Lastname').AsString := 'CWObjektteile '+s;
         FieldByName('Phone').AsString := s;
         GlobArray[k]:=StrToInt(FieldByName('Phone').AsString);
         post; {required!!!}
       end;
       close;
     end;
    t.active:=true;
    DataSource1.DataSet:=t;
  end;
```

Sort created table at runtime

```
procedure TBenchDlg.SortBtnClick(Sender: TObject);
  var
 j,k,tmp: integer;
begin
 for j:=1 to MaxArray-1 do
 begin
  for k:=1 to MaxArray-j do
   if GlobArray[k] < GlobArray[k+1] then
    begin
     { Swap GlobArray[k+] with GlobArray[k] ... }
     tmp := GlobArray[k];;
     GlobArray[k] := GlobArray[k+1];
     GlobArray[k+1]   := tmp;
  end;
 end;
     ListBox1.Clear;
     for j:= 1 to MaxArray  do
     Listbox1.Items.Add(IntToStr(GlobArray[j]));
end;
```

Delete created table at runtime

```
procedure TBenchDlg.DelBtnClick(Sender: TObject);
  var k,err: integer;
 s: string;
begin
 val(Edit1.Text,maxArray,err);
 t:=TTable.create(self);
     with t do
     begin
```

```
      DatabaseName := 'C:\DATEN\CALWIN2\CWtests'; {personal alias}
      TableName := 'pbench.db';
      open;
      edit;
      first;
      next;
      while not t.eof do
      begin
        delete;
      end;
      close;
    end;
   t.active:=true;
  DataSource1.DataSet:=t;
end;
```

Cell DBGridPosition in Runtime

```
var
         Col, Row: Integer;

       procedure TForm1.MyDBGridDrawDataCell(Sender: TObject; const Rect:
TRect;
         Field: TField; State: TGridDrawState);
       var
         RowHeight: Integer;
       begin
         if gdFocused in State then
         begin
           RowHeight := Rect.Bottom - Rect.Top;
           Row := (Rect.Top div RowHeight) - 1;
           Col := Field.Index;
         end;
       end;
```

Change the color of a grid cell in a TDBGrid

A: Enter the following code in the TDBGrid's OnDrawDataCell event:

```
Procedure TForm1.DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect;
  Field: TField; State: TGridDrawState);
begin
   If gdFocused in State then
      with (Sender as TDBGrid).Canvas do
      begin
         Brush.Color := clRed;
         FillRect(Rect);
         TextOut(Rect.Left, Rect.Top, Field.AsString);
      end;
end;
```

Set the Default drawing to true. With this, it only has to draw the highlighted cell. If you set DefaultDrawing to false, you must draw all the cells yourself with the canvas properties.

Create Indexes at runtime

Why is it that when I create a table using the TTable component's CreateTable method it creates the fields correctly but does not create the indexes even though I do a

```
    NewTable.IndexDefs.Assign(Table1.IndexDefs)?
```

A: This is the correct way to transfer the index definition to NewTable, however, the IndexDefs property of Table1 may not be up-to-date so you need to call the Update method of Table1's IndexDefs property prior to its assignment to NewTable like this example shows:

```
with NewTable do begin
  Active := False;
  DatabaseName := 'DBDEMOS';
  TableName := 'Temp';
  TableType := ttParadox;
  FieldDefs.Assign(Table1.FieldDefs);
  Table1.IndexDefs.Update;          { Do an update first }
  IndexDefs.Assign(Table1.IndexDefs);
  CreateTable;
end;
```

Scalable Screens

```
const
     ScreenHeight: integer = 800; {I designed my form in 800x600 mode.}
     ScreenWidth: integer = 600;

    procedure TForm1.FormCreate(Sender: TObject);
    var
      x, y: LongInt; {Integers will not not a large enough value.}

    begin
      form1.scaled := true;
      x := getSystemMetrics(SM_CXSCREEN);
      y := getSystemMetrics(SM_CYSCREEN);
      if (x <> ScreenHeight) or (y <> ScreenWidth) then
      with form1 do
      begin
          height := height * x DIV ScreenHeight;
          width := width * y DIV ScreenWidth;
          scaleBy(x, ScreenHeight);
        end;
    end;
```

Scale too the fonts:

```
type
  TFooClass = class(TControl); { needed to get at protected }
                               { font property }
var
  i: integer;
begin
  for i := ControlCount - 1 downto 0 do
    TFooClass(Controls[i]).Font.Size :=
        (NewFormWidth div OldFormWidth) *
        TFooClass(Controls[i]).Font.Size;
end;
```

Calculation with Date within Calculated Fields

```
procedure TForm1.Table1CalcFields(DataSet: TDataset);
```

```
var
  t1, t2: tDateTime;
begin
  table1d1.asDateTime := Date + 2; {or table1d1.value := date + 2;}
  table1d2.asDateTime := Date - 2;
  t1 := table1d1.asDateTime;
  t2 := table1d2.asDateTime;
  table1d3.asInteger := trunc(double(t1) - double(t2));
end;
```

Thousands Separator

```
function FormatNumber(l: longint): string;
var
  len, count: integer;
  s: string;
begin
  str(l, s);
  len := length(s);
  for count := ((len - 1) div 3) downto 1 do
    begin
      insert(',', s, len - (count * 3) + 1);
      len := len + 1;
    end;
  FormatNumber := s;
end;
```

If you are using Delphi, there is, of course, the easy way:

```
function FormatNumber(l: longint): string;

begin
  FormatNumber := FormatFloat('#,##0', StrToFloat(IntToStr(l)));
end;
```

Which memory model does Delphi use?

A: Delphi uses a mixed memory model, but it is very close to the "C" large model. The defaults are:

- Methods are far
- Procedures in an interface section are far
- Procedures only used in an implementation section are near
- Heap data and all pointers in general (including class instances) are far
- Global variables are near (DS based)
- Procedure parameters and local variables are near (SS based)
- Procedures declared FAR or EXPORT are far
- Virtual memory tables are far for the new class model and near for the old

This scheme has been used by Borland Pascal for a very long time. I find it flexible and efficient.

Since all public procedures, methods and pointers are 32bit already, Delphi32 won't have to change any of that. It's likely that Delphi32 will switch to 32bit addressing for the data and stack segments too, but that shouldn't affect any of your code either. What will affect it is the change of Integer from 16 to 32 bit.

{ This code came from Lloyd's help file! }

DLL Call

```
type
     TCallMeDll = function(a,b: Integer): string;

    var
     CallMeDll: TCallMeDll;
     FuncPtr: TFarProc;
     hDll: THandle;
     result: string;

    begin
     hDll:=LoadLibrary('Mytestdll.dll');
     FuncPtr:=GetProcAddress(hDLL,'CallMe');
     @CallMeDll:=FuncPtr;
     if @CallMeDll <> nil then
       result:=CallMeDll(4,5);
     FuncPtr:=nil;
     FreeLibrary(hDll);
    end;
```

Component Builder

TComponent - The base starting point for non-visual components.
TWinControl - The base starting point for components that need to have window handles.
TGraphicControl - A good starting point for visual components that don't need the overhead of a window handle. This class has a Paint method, that should be overridden, but no canvas.
TCustomControl - The most common starting point for visual components. This class has a Window handle, common events and properties, and most importantly a canvas with a Paint() method.

Was macht ein Konstruktor?

Wird der Konstruktor aufgerufen, dann gibt er eine Referenz auf eine neu allozierte und initialisierte Instanz des Klassentyps zurück.

Send the button a message to make it think it was

```
{ pressed again.  Doing so will cause this procedure to }
{ execute again, and the table will be opened without }
{ the MDX }
PostMessage(Button1.Handle, cn_Command, bn_Clicked, 0);
```

Books to Software Engineering

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

Books about Programming

Algorithms, by Robert Sedgewick; Addison-Wesley
Data Structures, Algorithms and Performance, by Derek Wood;

Addison-Wesley
Practical Data Structures in C++, by Bryan Flamig; Wiley

Also, the three volumes of The Art of Computer Programming by Knuth belong on every serious programmer's shelf, although they are starting to look their age as far as the programming examples go.

Benchmark with PowerBuilder

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

Power of Inheritance

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

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

Create Alias at Runtime

```
Database1.Params.Clear;
           Database1.Params.Add('PATH=C:\DELPHI\DEMOS\DATA');
          Table1.DatabaseName:= 'MyNewAlias';
          Table1.TableName:= 'CUSTOMER';
          Table1.Active:= True;
          DataSource1.DataSet:= Table1;
          DBGrid1.DataSource:= DataSource1;
        end;
```

«If as» in Inheritance

Die erste Behandlungsroutine gestattet dem Anwender, nur Elemente, die von Elementen des Typs TLabel gezogen wurden, abzulegen. Die zweite akzeptiert Elemente, die entweder von Elementen des Typs TLabel oder von beliebigen Nachkommen von TLabel gezogen wurden.

```
procedure TForm1.ListBox1DragOver(Sender, SOurce: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  if Sender.ClassType = TLabel then Accept := True;
end;

procedure TForm1.ListBox1DragOver(Sender, SOurce: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  if Sender is TLabel then Accept := True;
end;
```

Copy one TimeStamp from one File to Another

```
procedure CopyFileDate(const Source, Dest: String); Source and Dest are PathNames
var
  SourceHand, DestHand: word;
begin
  SourceHand := FileOpen(Source, fmOutput);      { open source file }
  DestHand := FileOpen(Dest, fmInput);           { open dest file }
  FileSetDate(DestHand, FileGetDate(SourceHand)); { get/set date }
  FileClose(SourceHand);                          { close source file }
  FileClose(DestHand);                            { close dest file }
end;
```

Change in an Editbox the Name and synchronise the tab

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

Getting the LineNumber from a Memo

```
procedure TMyForm.BitBtn1Click(Sender: TObject);
var
  iLine : Integer ;
begin
   iLine := Memo1.Perform(em_LineFromChar, $FFFF, 0);
   { Note: First line is zero }
   messageDlg('Line Number: ' + IntToStr(iLine), mtInformation,
              [mbOK], 0 ) ;
end;
```

SetRange Beispiel

```
begin
    with Table1 do begin
      SetRangeStart;
      FieldByName('LastName').AsString := 'S';
      SetRangeEnd;
      FieldByName('LastName').AsString := 'Szzz';
      ApplyRange;
    end;
  end;
```

Check Previous Instance

Testen ob das Program schon einmal gelden wurde

```
program Pprevins;
  uses
    WinTypes,
    WinProcs,
    SysUtils,
    Forms,
    Uprevins in 'UPREVINS.PAS' {Form1};

  {$R *.RES}
  type
    PHWND = ^HWND;
  function EnumFunc(Wnd:HWND; TargetWindow:PHWND): bool; export;
  var
    ClassName : array[0..30] of char;
  begin
    Result := true;
    if GetWindowWord(Wnd,GWW_HINSTANCE) = hPrevInst then
       begin
       GetClassName(Wnd,ClassName,30);
       if StrIComp(ClassName,'TApplication') = 0 then
          begin
          TargetWindow^ := Wnd;
          Result := false;
          end;
       end;
  end;

  procedure GotoPreviousInstance;

  var
    PrevInstWnd : HWND;
  begin
    PrevInstWnd := 0;
    EnumWindows(@EnumFunc,longint(@PrevInstWnd));
    if PrevInstWnd <> 0 then
       if IsIconic(PrevInstWnd) then
          ShowWindow(PrevInstWnd,SW_RESTORE)
       else
          BringWindowToTop(PrevInstWnd);
  end;

  begin
    if hPrevInst <> 0 then
       GotoPreviousInstance
    else
       begin
         Application.CreateForm(TForm1, Form1);
         Application.Run;
       end;
  end.
```

Menu remotable

```
  end.

begin
```

```
  PostMessage(Handle,wm_sysCommand,sc_keymenu, 0);
  PostMessage(Handle,wm_KeyDown,vk_Return, 0);
end;
```

All Ini Files in Delphi 1

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

Search in Querys

```
function SeqSearch(AQuery: TQuery; AField, AValue: String): Boolean;
  begin
    with AQuery do begin
      First;
      while (not Eof) and (not (FieldByName(AField).AsString = AValue)) do
        Next;
      SeqSearch := not Eof;
    end;
  end;
```

Pointer Arithmetic

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyArray: array[0..30] of char;
  b: ^char;
  i: integer;
begin
  StrCopy(MyArray, 'Lloyd is the greatest!'); {get something to point to}
  b := @MyArray; { assign the pointer to the memory location }
  for i := StrLen(MyArray) downto 0 do
  begin
    write(b^);   { write out the char at the current pointer location. }
    inc(b);      { point to the next byte (char is a byte!) in memory }
  end;

Difference Proof
var
  P1, P2 : ^LongInt;
  L : LongInt;
begin
  P1 := @L; { assign both pointers to the same place }
  P2 := @L;
  Inc(P2);  { Increment one }

{ Here we get the difference between the offset values of the
two pointers.  Since we originally pointed to the same place in
memmory, the result will tell us how much of a change occured
when we called Inc(). }

  L := Ofs(P2^) - Ofs(P1^); { L = 4; i.e. sizeof(longInt) }
end;
```

Move the Form by Timer

Die folgende Ereignisbehandlungsroutine reagiert auf Timer-Ereignisse mit einem Verschieben des aktiven Dialogelements um ein Pixel nach rechts:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ActiveControl.Left := ActiveControl.Left + 1;
end;
```

Call own Events

Die erste Frage, auf die Sie stoßen werden, wenn Sie Ihre eigenen Ereignisse definieren, ist, was das Ereignis auslöst. Um diesen Punkt brauchen Sie sich nicht zu kümmern, wenn Sie Standardereignisse verwenden. Die Antwort auf diese Frage ist bei einigen Ereignissen offensichtlich. Beispielsweise tritt ein Ereignis Maustaste gedrückt auf, wenn der Anwender die linke Taste der Maus drückt und Windows eine WM_LBUTTONDOWN-Botschaft an die Anwendung schickt. Nach dem Empfang der Botschaft ruft eine Komponente ihre Methode MouseDown auf, die ihrerseits den betreffenden Programmcode aufruft, den der Anwender mit dem Ereignis OnMouseDown verknüpft hat.

Beispiel

Die folgenden Methoden verwenden TControl, um die WM_LBUTTONDOWN-Botschaften von Windows zu verarbeiten. DoMouseDown ist eine private Implementierungsmethode, die eine generische Behandlung linker, rechter und mittlerer Maustastenklicks zur Verfügung stellt. Sie übersetzt die Parameter der Windows-Botschaft in Werte für die Methode MouseDown.

```
type
  TControl = class(TComponent)
  private
    FOnMouseDown: TMouseEvent;
    procedure DoMouseDown(var Message: TWMMouse; Button: TMouseButton;
      Shift: TShiftState);
    procedure WMLButtonDown(var Message: TWMLButtonDown); message
WM_LBUTTONDOWN;
  protected
    procedure MouseDown(Button: TMouseButton; Shift: TShiftState;
      X, Y: Integer); dynamic;
  end;
...
procedure TControl.MouseDown(Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Assigned(FOnMouseDown) then
    FOnMouseDown(Self, Button, Shift, X, Y);     { Behandlungsroutine aufrufen,
          falls vorhanden }

end;

procedure TControl.DoMouseDown(var Message: TWMMouse; Button: TMouseButton;
  Shift: TShiftState);
begin
  with Message do
    MouseDown(Button, KeysToShiftState(Keys) + Shift, XPos, YPos);     { Dynamische
Methode
```

```
            aufrufen }
end;

procedure TControl.WMLButtonDown(var Message: TWMLButtonDown);
begin
  inherited;        { Standardbehandlungsroutine durchführen }
  if csCaptureMouse in ControlStyle then MouseCapture := True;
  if csClickEvents in ControlStyle then Include(FControlState, csClicked);
  DoMouseDown(Message, mbLeft, []);  { generische Methode Maustaste gedrückt
aufrufen }
end;
```

How select a specific field on a TDBGrid to get focus?

A: Using this code:

```
DBGrid1.SelectedField := Table1SomeField;
DBGrid1.SetFocus;
```

Datenexport aus einer Datenbank-Tabelle in eine ASCII-Datei.

```
procedure TMyTable.ExportToASCII;

var
  I: Integer;
  Dlg: TSaveDialog;
  ASCIIFile: TextFile;
  Res: Boolean;

begin
  if Active then
    if (FieldCount > 0) and (RecordCount > 0) then
      begin
        Dlg := TSaveDialog.Create(Application);
        Dlg.FileName := FASCIIFileName;
        Dlg.Filter := 'ASCII-Dateien (*.asc)|*.asc';
        Dlg.Options := Dlg.Options+[ofPathMustExist,
          ofOverwritePrompt, ofHideReadOnly];
        Dlg.Title := 'Daten in ASCII-Datei exportieren';
        try
          Res := Dlg.Execute;
          if Res then
            FASCIIFileName := Dlg.FileName;
        finally
          Dlg.Free;
        end;
        if Res then
          begin
            AssignFile(ASCIIFile, FASCIIFileName);
            Rewrite(ASCIIFile);
            First;
            if FASCIIFieldNames then
              begin
                for I := 0 to FieldCount-1 do
                  begin
                    Write(ASCIIFile, Fields[I].FieldName);
                    if I <> FieldCount-1 then
                      Write(ASCIIFile, FASCIISeparator);
                  end;
                Write(ASCIIFile, #13#10);
```

```
       end;
      while not EOF do
       begin
        for I := 0 to FieldCount-1 do
         begin
          Write(ASCIIFile, Fields[I].Text);
          if I <> FieldCount-1 then
           Write(ASCIIFile, FASCIISeparator);
         end;
        Next;
        if not EOF then
         Write(ASCIIFile, #13#10);
       end;
      CloseFile(ASCIIFile);
      if IOResult <> 0 then
       MessageDlg('Fehler beim Erstellen oder Schreiben '+
        'in die ASCII-Datei', mtError, [mbOK], 0);
     end;
   end
  else
   MessageDlg('Es sind keine Tabellendaten zu exportieren.',
    mtInformation, [mbOK], 0)
 else
  MessageDlg('Datenbank-Tabelle muß geöffnet sein, damit Daten '+
   'ins ASCII-Format exportiert werden können.', mtError,
   [mbOK], 0);
end;
```

Check Connection of Database.

(zum Beispiel, ob ein Zugriff auf die Daten möglich ist) und einen Statuswert zurückzugeben (True oder False)

```
function TBDEDirect.CheckDatabase: Boolean;

var
 DS: TDataSource;

begin
 Result := False;
 DS := GetDataSource;
 if DS = nil then
  begin
   MessageDlg('Die Anbindung an ein Datenquell-Element fehlt. '+
    'Stellen Sie die Eigenschaft DataSource entsprechend ein.',
    mtError, [mbOK], 0);
   Exit;
  end;
 if DS.DataSet = nil then
  begin
   MessageDlg('Zugriff auf Datenbank nicht möglich.', mtError,
    [mbOK], 0);
   Exit;
  end;
 if TDBDataSet(DS.DataSet).Database = nil then
  begin
   MessageDlg('Zugriff auf Datenbank nicht möglich.', mtError,
    [mbOK], 0);
   Exit;
  end;
 if TDBDataSet(DS.DataSet).Database.Handle = nil then
```

```
    begin
      MessageDlg('Datenbank-Handle nicht verfügbar.', mtError,
        [mbOK], 0);
      Exit;
    end;
  if DS.DataSet.Handle = nil then
    begin
      MessageDlg('Cursor-Handle nicht verfügbar.', mtError,
        [mbOK], 0);
      Exit;
    end;
  Result := True;
end;
```

Grösse von Forms steuern

Dieses kleine Beispiel zeigt Ihnen, wie die Größe von Formularen beschränkt werden kann. Wenn die Eigenschaft BorderStyle den Wert bsSizeable besitzt, kann das Formular stufenlos vergrößert und verkleinert werden. Meist ist es jedoch nötig, daß das Formular eine Mindestgröße besitzt, um noch alle nötigen Informationen aufnehmen zu können. Die folgende Routine zeigt, wie die Werte für die Mindestgröße eines Formulars angegeben werden können.

```
type
  TForm1 = class(TForm)
    procedure wmGetMinMaxInfo(var Msg : TMessage); message wm_GetMinMaxInfo;

procedure TForm1.wmGetMinMaxInfo(var Msg : TMessage);

begin
  PMinMaxInfo(Msg.lParam)^.ptMinTrackSize.X := 600;
  PMinMaxInfo(Msg.lParam)^.ptMinTrackSize.Y := 350;
end;
```

Get Execution-Path

Man bedient sich der Funktion "ExtractFilePath", die Laufwerk und Pfad, sowie einen Backslash zurückgibt.

```
Verzeichnis := ExtractFilePath(Application.ExeName);
```

oder als SchnellMethode **.**

Logical Size of Reocord

Mit Hilfe der Funktion TBDEDirect.GetLogicalRecSize wird die logische Größe des aktuellen Datensatzes ermittelt.

```
function TBDEDirect.GetCursorProps: CurProps;

var
  CP: CurProps;
  Res: DBIResult;

begin
  FillChar(CP, SizeOf(CP), #0);
  Result := CP;
  if CheckDatabase then
    begin
      Res := DbiGetCursorProps(FDataLink.DataSource.DataSet.Handle, CP);
      if Res = 0 then
        Result := CP
    else
      Check(Res);
  end;
end;

function TBDEDirect.GetLogicalRecSize: Word;

begin
  Result := GetCursorProps.iRecSize;
end;
```

OLE 2 Automation

Delphi 2 unterstützt voll die OLE 2.0 Automation. Eine Anwendung (OLE-Client) kann einen sogenannten OLE-Server aufrufen, der die Aktionen dieser Anwendung überwacht.
Hier ist ein Code-Beispiel, in dem die Ergebnisse einer Datenbankabfrage (zusammengefaßt in einer Tabelle) in WinWord eingefügt werden:

```
procedure TForm1.InsertBtnClick(Sender: TObject);
var
  WinWord: Variant;
  S: string;
  L: Integer;
begin
  { Aufbau der Verbindung zu dem Server in WinWord und Starten }
  { der Datenbankabfrage                                       }
  WinWord := CreateOleObject('Word.Basic');
  with Query1 do
  begin
    Close;
    Params[0].Text := Edit1.Text;
    Open;
    try
      First;
      L := 0;
      while not EOF do
      { Speichern des Abfrageergebnisses in dem String S }
      begin
        S := S + Query1Company.AsString + ',' +
          Query1OrderNo.AsString + ',' +
          Query1SaleDate.AsString + #13;
        Inc(L);
        Next;
      end;
      { OLE-Automation wird benutzt, um S in WinWord einzufügen}
      WinWord.Insert(S);
      WinWord.LineUp(L, 1);
      WinWord.TextToTable(ConvertFrom := 2, NumColumns := 3);
    finally
      Close;
    end;
  end;
end;
```

Get physicalRecSize

Die Funktion TBDEDirect.GetPhysicalRecSize liefert die physikalische (tatsächliche) Größe eines Datensatzes zurück.

```
function TBDEDirect.GetCursorProps: CurProps;

var
  CP: CurProps;
  Res: DBIResult;

begin
  FillChar(CP, SizeOf(CP), #0);
  Result := CP;
  if CheckDatabase then
    begin
      Res := DbiGetCursorProps(FDataLink.DataSource.DataSet.Handle, CP);
      if Res = 0 then
        Result := CP
      else
        Check(Res);
    end;
end;

function TBDEDirect.GetPhysicalRecSize: Word;

begin
  Result := GetCursorProps.iRecBufSize;
end;
```

Get Numbers of Records/Seq

mit Hilfe der folgenden Funktion wird die Anzahl der Datensätze ermittelt. Um dies zu erreichen, bedient man sich der BDE-Funktion DbiGetRecordCount.

```
function TBDEDirect.GetRecordCount: LongInt;

var
  Count: LongInt;
  Res: DBIResult;

begin
  Result := -1;
  if CheckDatabase then
    begin
      Res := DbiGetRecordCount(FDataLink.DataSource.DataSet.Handle,
        Count);
      if Res = 0 then
        Result := Count
      else
        Check(Res);
    end;
end;

uses DbiProcs, DbiTypes;

procedure TForm1.DataSource1DataChange(Sender: TObject; Field: TField);
var
  recNo: LongInt;
begin
  if Table1.State = dsInactive then
     begin
        MessageDlg('Table must be active.', mtError, [mbOK], 0);
        Exit;
     end;
  Table1.UpdateCursorPos;
  dbiGetSeqNo(Table1.Handle, recNo);
  Label1.Caption := 'Record No: ' + IntToStr(recNo);
end;
```

Check SQL-Query

Dier folgende Routine zeigt eine Möglichkeit, wie man eine SQL-Abfrage überprüfen kann.

```
procedure TMyQuery.CheckSQL;

begin
  if not Active then
    begin
      if (DatabaseName = '') and (DataSource = nil) then
        begin
          MessageDlg('Kein Datenbankzugriff. Stellen Sie die '+
            'Eigenschaft DatabaseName bzw. DataSource '+
            'entsprechend ein.', mtError, [mbOK], 0);
          Exit;
        end;
      if SQL.Count = 0 then
        begin
          MessageDlg('Kein SQL-Text', mtError, [mbOK], 0);
          Exit;
        end;
      try
        OpenCursor;
      except
        CloseCursor;
        MessageDlg('Fehler in dem SQL-Text oder Datenbank-Fehler',
          mtError, [mbOK], 0);
        Exit;
      end;
      MessageDlg('SQL-Text korrekt', mtInformation, [mbOK], 0);
      CloseCursor;
    end
  else
    MessageDlg('SQL-Abfrage bereits aktiv', mtInformation, [mbOK], 0);
end;
```

Tabulatoren zur Laufzeit setzen

Hierzu muß man das Property "WantTabs" auf "True" setzen. Möchte man auch noch die Tabweite setzen, so muß man die API-Funktion SendMessage aufrufen. Als Parameter erwartet die Funktion unter anderem einen Pointer auf ein Array des Typs Word, in dem die einzelnen Positionen in Bildschirmeinheiten gespeichert sein müssen. Zum Beispiel hier der Aufruf für 2 Tabstops:

```
procedure TForm1.FormCreate(Sender:TObject);

const
  Tabs: array[0..1] of Word = (4, 8);

begin
  SendMessage(Memo1.Handle, EM_SetTabStops, 2, LongInt(@Tabs));
end;
```

Bitmap size drawing

Hierzu folgende Unit:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
  end;

var
  Form1: TForm1;
  Bitmap: TBitmap;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);

begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('C:\WINDOWS\WINLOGO.BMP');
end;

procedure TForm1.FormPaint(Sender: TObject);

var
  x, y, w, h: LongInt;

begin
  with Bitmap do
    begin
      w := Width;
      h := Height;
    end;
  y := 0;
  while y < Height do
    begin
      x := 0;
      while x < Width do
        begin
          Canvas.Draw(x, y, Bitmap);
          Inx(x, w);
        end;
      Inc(y, h);
  end;
end;

end.
```

Schutz von Methoden

Alle Teile von Objekten, einschließlich Felder, Methoden und Eigenschaften, können verschiedenen Schutzklassen angehören, wie in Den Zugriff kontrollieren beschrieben. Die Wahl der richtigen Schutzklasse für Methoden ist einfach.
Allgemein gilt, daß Methoden, die Sie für Ihre Komponenten schreiben, entweder public oder protected sind. Die Ausnahme von dieser Regel bilden Methoden, die Eigenschaften implementieren; diese sollten immer private sein. Unter anderen Umständen ist es sehr selten erforderlich, eine Methode als private zu deklarieren, es sein denn, sie ist wirklich spezifisch für genau diesen Komponententyp, so daß nicht einmal daraus abgeleitete Komponenten Zugang zu der Methode haben sollten.

Hinweis: Es gibt keinen Grund, Methoden (im Unterschied zu Ereignisbehandlungsroutinen) als published zu deklarieren. Für den Benutzer würden sie genauso aussehen wie bei einer Deklaration als public.

Nachricht an Button für z.B. ein CBT-Demo

```
procedure TDemo2Form.MaxTimer(const delay: byte);
var StartTime: real;
begin
  StartTime:=Time;
  Repeat
    WaitMessage;
  Until Time > StartTime + delay*(1/24/60/60);
end;


procedure TDemo2Form.selfButtonClick(Sender: TObject);
begin
    MaxTimer(2);

  PostMessage(Button1.Handle, cn_Command, bn_Clicked, 0);
   SuchenForm.ShowModal;
   MaxTimer(2);
    SuchenForm.SearchEdit.Text:='Tewi';

    MaxTimer(3);
```

Draw in a DBGrid

```
begin
    if (Field.FieldName = dbcheckbox1.DataField) then
    begin
      if MWKmwX12.AsBoolean then
        mwgrid.Canvas.Draw(Rect.Left,Rect.Top, ImageTrue.Picture.Bitmap)
      else
        mwgrid.Canvas.Draw(Rect.Left,Rect.Top, ImageFalse.Picture.Bitmap)
      {  DBGrid1.Canvas.StretchDraw(Rect, ImageFalse.Picture.Bitmap); }
    end
    { Do bit map dwawing if you want}
   { mwgrid.Canvas.FillRect(Rect); }
  end;
```

Listen mit Objekten verwalten

```
type
  TMyClass = class
    MyString: string;
    constructor Create(S: string);
  end;
```

```
constructor TMyClass.Create(S: string);
begin
  MyString := S;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  MyList: TList;
  MyObject, SameObject: TMyClass;
begin
try
  MyList := TList.Create;                                    { Liste erzeugen }
  try
    MyObject := TMyClass.Create('Semper Fidelis!');         { Klasseninstanz erzeugen }
      MyList.Add(MyObject);                          { Instanz zu Liste hinzufügen }
      SameObject := TMyClass(MyList.Items[0]);      { erstes Element in Liste abfragen }
      MessageDlg(SameObject.MyString, mtInformation, [mbOk], 0);        { und anzeigen }
    finally
      MyObject.Free;                              { Aufräumen nicht vergessen! }
 end;
finally
    MyList.Free;
end;
end;
```

Button auf enable abfragen

25Q)Is there any way to determine if a particular button on a TDBNavigator control is enabled?

25A)Try:

```
type
  TDBNavCracker = class(TDBnavigator);
 ...
if TDBNavCracker(DBNavigator1).Buttons[nbEdit].Enabled
then...;
```

Probleme Update 1.02

Dieser Abschnitt stellt eine allgemeine Liste mit Problemen, die in diesem Release beseitigt wurden.

ReportSmith

o In der ReportSmith-Datei READRS.TXT finden Sie eine Liste mit den behobenen Problemen

Delphi

o Probleme im Delphi-Online-System wurden beseitigt
o Die Kompatibilität zu Windows 95 für MDI (z.B. new child) und OLE2 (z.B. insert object) wurde verbessert

o Probleme bei der IDE-Debugger-Kompatibilität zu Windows NT wurden behoben
o Große Änderungen an der OLE2 API Unit (Siehe \DELPHI\DOC\OLE2.INT)
o Unit-Versionsproblem in der Datei DLIB.EXE wurde behoben
o Das Problem im Browser, das beim Doppelklicken auf einen Verweis auf eine geschlossene PAS-Datei auftrat, wurde beseitigt
o Das Problem bei Auswahl des Befehls "Optionen|Bibliothek neu generieren", wenn das aktuelle Projekt über ein aktives Datenset

verfügt, wurde beseitigt

- o Problem mit der Tastenkombination ALT+TAB in der Rastersteuerung behoben
- o TForm.DefineProperty ruft nun seine vererbten Methoden auf
- o Unterstützung für benutzereigene Zeichnungen in TOutline
- o DBImage.CutToClipboard aktualisiert die Zwischenablage nun korrekt
- o TDataSource.OnDataChange; ungültiger Zeiger im Feldparameter korrigiert.
- o Verschiedene Probleme in der Demo beseitigt

Local InterBase

- o Die Leistung bei Indexerstellung und SQL-Anforderung,

die sich auf Sortiervorgänge von Daten beziehen, deren Größe den beim Erstellen definierten Datenbank-Cache überschreitet, wurde verbessert.

- o Verbesserte E/A-Diagnose und Fehlermeldungen durch den Server
- o Die Local InterBase-Engine wurde verbessert, so daß komplexe Anforderungen, bei denen mehr als 500 Spalten referenziert werden, wodurch die internen Puffer die Grenze von 64 KB überschreiten, verarbeitet werden können
- o Verbesserte Leistung bei Local InterBase-Anforderungen, die mehr

als 500 Spalten referenzieren

- o Die interne Anforderung der Loclal InterBase, durch die die Seitengröße auf 1024 festgelgt war, wurde beseitigt. Sie können nun eine Datenbank mit jeder gültigen InterPage-Seitengröße erstellen und einsetzen
- o Fehler bei numerischen Überläufen werden nun von der Local InterBase abgefangen und richtig in der Client-Anwendung angezeigt
- o Inkompatibilitätsprobleme bei der Verwendung von DashBoard (Starfish Inc.) zum Anzeigen von Benutzern in der InterBase-

Sicherheitsdatenbank (Task|Benutzersicherheit) mit dem InterBase Server Manager-Tool wurden beseitigt

- o Probleme beim Bereinigen, die auftraten, wenn das Windows Interactive SQL-Tool während einer Datenbankvalidierung (Task|Datenbankvalidierung) im InterBase Server Manager beendet wurde, wurden beseitigt

BDE

Diese Version der Borland Datenbank-Engine entspricht der Version, die im neuen Visual dBase 5.5 verwendet wird.

- o TQuery und TTable wurden so geändert, daß sie nun Oracle-

Synonyme unterstützen. Synonyme können nun angezeigt werden, indem der Name in die Tabelleneigenschaft Name eingegeben wird

- o Das Bearbeiten einer geöffneten Paradox-Tabelle in Delphi simultan mit referentiellen Integritätsregeln ist nun möglich
- o Problem bei der Unterstützung von gespeicherten Prozeduren, die über String-Parameter verfügen, wurde beseitigt
- o Probleme in bezug auf den Zugriff auf Tabellen über ein Lantastic 6.0 Netzwerk wurden behoben

- o SQL-Abfragen unterstützen nun mehr als 9 Parameter
- o Probleme in bezug auf das Öffnen von Access 2.0 Dateien über ODBC wurden beseitigt
- o Die Probleme bei der DBD wurden behoben. Es können nun

Watcom 4.0 Tabellen geöffnet werden

- o Das 32 K Speicherloch, das Probleme bei dynamischen Abfrageergebnissen und lokalen Abfragen, in denen Joins verwendet werden, hervorgerufen hat, wurde beseitigt
- o Die Sybase Forceindex-Funktion wird nun unterstützt
- o Informix 5.X wird nun unterstützt

FocusControl mit Labels

Diese Code-Zeilen zeigen eine Textzeile in einem Label auf dem Formular an und assoziieren das Label mit einem Editierfenster. Beachten Sie, daß der Label-Titel ein Tastenkürzel beinhaltet. Betätigt der Benutzer Alt+N, so erhält das Editierfenster den Fokus:

```
Label1.Caption := '&Name';
Label1.FocusControl := Edit1;
```

Bei diesem Beispiel sollten das Label und das Editierfenster dicht beieinander plaziert werden, um dem Benutzer zu zeigen, daß er Text in das Editierfenster eintragen kann.

Deklaration

```
function FocusControl;
```

Beschreibung

Diese Methode setzt den Fokus in einem Formular auf die erste datensensitive Komponente, die mit einem TField assoziiert ist. Man benutzt diese Methode zur satzbezogenen Prüfung (z.B. im Ereignis BeforePost), da ein Feld möglicherweise geprüft wird, ob die assoziierten datensensitiven Komponenten den Fokus haben oder nicht.

Delphi BonMot des Jahres

Man kann virtuelle oder dynamische Methoden als wirklich abstrakt deklarieren. Zum Deklarieren einer Methode als wirklich abstrakt benutzt man die Anweisung abstract. Das bedeutet, daß man die Methode nicht implementiert, sondern nur die Überschrift deklariert.

Modale Fenster immer oben rechts

```
PROCEDURE tDBInfoWindow.FormCreate(Sender: tObject);
  BEGIN
    Left := Screen.Width - Width - 5;
    Top  := 5;
  END;
```

OnDataChange

```
PROCEDURE tMainWindow.DataSource1DataChange(
                              Sender: tObject;
                              Field : tField);
  BEGIN
    TRY
      RecPos1.UpdateLabel;
 EXCEPT
    END;
  END;
```

Control zur Laufzeit verschieben

```
with Btn1 do
  begin
    parent:=self;
    top:=toplace;
    left:=DBNavi.width + interspace + 7;
    visible:=false;
  end;

with Btn2 do
  begin
    parent:=self;
    top:=toplace;
    left:=Btn1.left + Btn1.width + interspace;
  end;
```

Daten von Zeilen und Spalten in DBGrid berechnen

```
procedure TForm1.CustCalcFields(DataSet: TDataset);
begin
  CusttotalTax.value:= CustCustNo.value + 6;
end;

procedure TForm1.Calcul;
var tempo: real;
begin
  with  CUST do
  begin
    first;
    tempo:=0;
    while not EOF do
    begin
      tempo:= tempo + CustTotalTax.value;
      next;
    end;
    totalLbl.Caption:=floatToStr(tempo);
  end;
end;
```

Drag and Drop von DB auf ein Feld

```
procedure TForm1.RangeStartDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept:= Source is TDBGrid;
end;

procedure TForm1.RangeStartDragDrop(Sender, Source: TObject; X,
  Y: Integer);
begin
   messageBeep(0);
   with Source as TDBGrid do
   begin
     if (sender as TEdit).tag = 1 then
     begin
```

```
      with rangeStart do
      text := intToStr((Source as TDBGrid).SelectedField.asInteger);
    end else
      with rangeEnd do
      text:= IntToStr((Source as TDBGrid).SelectedField.asInteger);
  end;

end;
```

Send own messages to the queue

```
const
  um_UpdatePosition = WM_USER + 0;

type
  TForm1 = class(TForm)
    ...
    procedure memEditKeyDown(...);
  protected
    procedure UMUpdatePosition(var message: TMessage);  message
                                              um_UpdatePosition;
    ...
  end;

implementation

procedure TForm1.memEditKeyDown(...);
begin
  PostMessage(Handle, um_UpdatePosition, 0, 0);  {put message on queue}
end;

procedure TForm1.UMUpdatePosition(var message : TMessage);
begin
 do something
end
```

Add a component at runtime to Notebook/TabbedNotebook

```
procedure AddButton(tabNotebook : TTabbedNotebook);
var
  tabpage : TTabPage;
  button  : TButton;
begin
  with tabNotebook do
    tabpage := TTabPage(Pages.Objects[PageIndex]);
  button := TButton.Create(tabpage);
  try
    with button do begin
      Parent := tabpage;
      Left   := Random(tabpage.ClientWidth - Width);
      Top    := Random(tabpage.ClientHeight - Height);
    end;
  except
    button.Free;
  end;
end;
```

Iterate and count the controls with logger

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
logList: TStringList;
begin
logList: TStringList.create;
  for I := 0 to ComponentCount -1 do
 {    if Components[I] is TButton then
       TButton(Components[I]).Font.Name := 'Courier';
 Edit1.Text := IntToStr(ComponentCount) + ' components';}
  begin
     myList.Add(intToStr(i) + ' count');
     myList.Add(components[i].name);
   end;
myList.SaveToFile('LOGLIST.TXT');
end;
```

Routine zum Cracken eines Files

```
procedure TForm1.Button1Click(Sender: TObject);
 var   PgIndex,i : Integer;
 myList: TStringlist;
 openDlg :TOpenDialog;
 saveDlg :TSaveDialog;
  openF, saveF: TextFile;
  Ch: Char;
begin
  myList:=TStringList.Create;
  openDlg:=TOpenDialog.Create(self);
  saveDlg:=TSaveDialog.Create(self);
  openDlg.Filter := 'Text files (*.*)|*.*';
  saveDlg.Filter := 'Crackfiles (*.*)|*.*';
  filterset:=[0..34, 65..67];

  if z=4 then            {debug}
  if openDlg.Execute then         { Display Open dialog box }
  begin
    AssignFile(openF, openDlg.FileName);   { File selected in dialog box }
    Reset(openF);
    if saveDlg.Execute then
    begin
      AssignFile(saveF, saveDlg.fileName);
      Rewrite(saveF);
      while not EOF(openF) do
      begin
        Read(openF, Ch);                { Read the first line out of the file }
        if (ord(Ch) in filterSet) then
        begin
          Memo1.Lines.Add(Ch);         { Put string in a TMemo control }
          Write(saveF, Ch);
        end;
      end;
     CloseFile(saveF);
    end; {save}
   CloseFile(openF);
  end; {open}
end;
```

All the loaded Modules in D

DBD
DBD.EXE 166496
DBDQBE.DLL 95952
DBDSQL.DLL 42496
DBDUTILS.DLL 108048
DBCOEDIT.DLL 30656
DBDCREAT.DLL 269424
DBDVIEW.DLL 246096
DBDSRV.DLL 675560

BIN
COMPLIB.DCL
EXPTDEMO.DLL
DELPHI.EXE
DELPHIED.DLL
DELPHIKB.DLL
PASDBK16.DLL
W8LOSS.DLL

IDAPI
IDAPI01.DLL 395824
IDR10009.DLL 20800
ILD01.DLL 47616
IDBAT01.DLL 105306
IDPDX01.DLL 248336

XOR Encryption

```
function Encrypt(const S: String; Key: Word): String;
var
  I: byte;
begin
  Result[0] := S[0];
  for I := 1 to Length(S) do begin
    Result[I] := char(byte(S[I]) xor (Key shr 8));
    Key := (byte(Result[I]) + Key) * C1 + C2;
  end;
end;

function Decrypt(const S: String; Key: Word): String;
var
  I: byte;
begin
  Result[0] := S[0];
  for I := 1 to Length(S) do begin
    Result[I] := char(byte(S[I]) xor (Key shr 8));
    Key := (byte(S[I]) + Key) * C1 + C2;
  end;
end;

procedure TTabTest.EncBtnClick(Sender: TObject);
begin
  S:= Edit2.text;
  S := Encrypt(S,12345);
  Edit2.text:=S;
  S := Decrypt(S,12345);
  Edit3.text:=S;

end;
```

Open Arrays

```
type
  TAnArray = array[0..200] of longint;
  TAnotherArray = array[-4..6] of longint;

function TTabTest.CalcSum(AnArray: array of longint): longint;
var
  Sum: integer;
  i: integer;
begin
  Sum := 0;
  with ListBox1.Items do
  begin
    Add('Unterer Index: ' + IntToStr(Low(AnArray)));
    Add('Oberer Index: ' + IntToStr(High(AnArray)));
  end;
    for i:=Low(AnArray) to High(AnArray) do
    Sum := Sum + AnArray[i];
  CalcSum := Sum;
end;


procedure TTabTest.InitArray(var AnArray: array of longint);
{ das ist die Form um Parameter als offene Arrays zu deklarieren }
var
  i:integer;
begin
  { innerhalb der Prozedur wir das Array als 0-basiert angenommen }
  {koennte auch sein for i:=0 to, beginnt immer bei 0}
  for i:=Low(AnArray)  to High(AnArray) do
    AnArray[i] := i;
end;


procedure TTabTest.opArrayBtnClick(Sender: TObject);
var
  AnArray: TAnArray;
  AnotherArray: TAnotherArray;
begin
  ListBox1.Clear;
  InitArray(AnArray);
  InitArray(AnotherArray);
  With ListBox1.Items do
  begin
    Add('Summe: ' + IntToStr(CalcSum(AnArray)));
    Add('Summe: ' + IntToStr(CalcSum(AnotherArray)));
  end;
end;
```

API intercept

```
procedure WMPaint(var Message: TWMPaint); message WM_Paint;

procedure TDemo2Form.WMPaint(var Message: TWMPaint);
    begin
       waitmessage;
       messagebeep(0);
       canvas.Textout(random(50),random(50) ,'my win hook is haunted');
       inherited;
       {enablewindow(newTexbtn.handle, false);}
       {have your way with the component}
    end;
```

SQL Search Dialog with Substitutionsparameter

```
procedure TDemo2Form.SuchenClick(Sender: TObject);
var searchresult,
    searchString: string[30];
const it: byte=0;
begin
  searchString:=' Suchen Sie was';
  with Query1 do
  begin
    disableControls;
    try
      active:=false;
      Close;
      SQL.Clear;
      Query1.SQL.Add('Select * from pbench where FIRSTNAME=:Name');
      if InputQuery('search','prompt',searchString) then
      begin
        Params[0].AsString:=searchString;
        Open;
        active:=true;
      end;
    finally
    enableControls;
  end;
end;
```

Scrollbar with Key Control

```
procedure TDemo2Form.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
   const  PageDelta = 10;
   if Key = VK_ESCAPE then Close;
     With VertScrollbar do
       if Key = VK_NEXT then  Position := Position+PageDelta
       else if Key = VK_PRIOR then Position := Position-PageDelta;
   end;
```

Nodes and Node Stores

We all know that these classical data structures are generally implemented with nodes. For a doubly linked list a node is usually represented by a structure of the form:

```
  PMyNode = ^TMyNode;
  TMyNode = record
    Next : PMyNode;     {Link to next node in the chain}
    Prev : PMyNode;     {Link to previous node in the chain}
    Data : SomeRecord;
  end;
```

Debugging and Errors and Exceptions

When I started writing this unit I had several goals, but there were two which seemed incompatible: it had to be fast and it had to have lots of checks built in to trap any errors that might occur.

This second goal is further complicated because there are broadly two types of error that could occur: an error due to a programming mistake and errors due to some run-time problem. This is necessarily a wishy-washy definition, but generally the run-time problems would be things like running out of memory whilst adding a data object, and the programming mistakes would be things like trying to pop a data object from an empty stack. Another way to view this would be to define programming mistakes as being those things which would apply to every machine, whereas the run-time problems would vary from user to user and from machine to machine. Normal testing should identify programming mistakes, whereas the other type of error are exceptions to the norm.

Assert in c (Precompiler) in Pascal

An example should make this clear. The TStack object has a method called Pop to remove the topmost data object from the stack. If the stack is empty, I count calling Pop as a programming mistake: you really should check for the stack being empty in your program prior to calling Pop. Of course Pop could have an if statement within it that did this check for you, but in the *majority* of cases the stack won't be empty when Pop is called and in the *majority* of cases when you use Pop, you'll have some kind of loop in your program which is continually checking whether the stack is empty or not anyway. In my mind having a check for an empty stack within Pop is safe but slow. So, instead, Pop has a call to an Assert procedure at the start (activated by the DEBUG compiler define) that checks to see whether the stack is empty. Here is the code for Pop:

```
function TStack.Pop : pointer;
  var
    Node : PNode;
  begin
    {$IFDEF DEBUG}
    Assert(not IsEmpty, ascEmptyPop);
    {$ENDIF}
    Node := Head^.Link;
    Head^.Link := Node^.Link;
    Pop := Node^.Data;
    acDisposeNode(Node);
  end;
```

As you see, if DEBUG is set the Assert procedure checks whether the stack is empty first, if not it executes the code that pops the data object off the stack. If the stack is empty an EEZAssertionError exception is raised (the constant ascEmptyPop is a string code for a stringtable resource). If DEBUG is not set the code runs at full speed.

TPriorityQueue (Heap)

=============

A priority queue is much like an ordinary queue, except that the smallest data object in the queue will be popped first (rather than the 'oldest'). Another name for a priority queue is a heap (not to be confused with the Pascal heap where memory blocks are allocated and freed). As it imposes a sort order on the data objects, you must override the Compare function.
Priority Queues basieren intern auf einer Heap-Datenstruktur. Ein Heap ist ein binärer Baum, bei dem die zwei untergeordneten Knoten kleiner oder gleich ihrem übergeordneten (Eltern)-Knoten sind. Diese Art von Baum lässt sich sehr einfach in einem Array abbilden.

How do I set focus on a specific field on a TDBGrid?

A:

```
DBGrid1.SelectedField := Table1Field1;
DBGrid1.SetFocus;
```

{ This code came from Lloyd's help file! }

Navigator control use

Q: I have a form that uses several TDBGrids. It has only one navigator control. How do I write it so that I can use the navigator control so that it works with whatever grid is active?

A: Use this line in the Enter event of each grid:

```
TDBNavigator1.dataSource := (sender as TDBGrid).dataSource;
```

{ This code came from Lloyd's help file! }

How can I use a TList to hold variables?

A:

```
implementation
type
  pLongInt = ^LongInt;

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  t: tlist;
  l: longint;
begin
  t := tlist.create;
  l := 123;
  t.add(@l);
  caption := IntToStr(pLongInt(t.items[0])^);
  t.free;
end;
```

2. Variante

```
var
  ptrList: tlist;
  lptr :^longint;
begin
  ptrList := tlist.create;
  new(lptr);
  lptr^ := 123456;
  ptrList.add(lptr);
  caption := IntToStr(plongint(ptrList.items[0])^); {typecasting}
  ptrList.free;
  dispose(lptr);
end;


{ This code came from Lloyd's help file! }
```

Actual and formal parameter with pointers

```
{The following example modifies a copy of the parameter.}
procedure ValueEx (X :Integer);
var
  ptr := ^integer;       Achtung Fehler in der OnlineHilfe unter the @ Operator > pointers
begin
  ptr := @X;
  writeln(Ptr^);
  Ptr^ := 15;
end;

var
  Fred : integer;
begin
  Fred := 10;
  ValueEx (Fred);
  Writeln (Fred);    {10}
end.

{The following example modifies the actual parameter.}
procedure VarEx(var Y : integer);
var Ptr = ^integer;
begin
  Ptr := @Y;
  writeln (Ptr^);
  Ptr^ := 15;
end;

var Fred : integer;

begin
  Fred := 10;
  VarEx (Fred);
  writeln (Fred);    {15}
end.
```

Iterate through tabbed notebook pages to see each object?

A: Here is a procedure that will iterate through all tabbed notebook pages and add the object name and type under the page's name in an outline.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  cmpnts, pg: word;
  MyPageObj: TObject;
  OutlineIdx: longint;
begin
  for pg := 0 to TabbedNotebook1.pages.count - 1 do begin
    MyPageObj := TabbedNotebook1.pages.objects[pg];
    OutlineIdx := outline1.add(0, TabbedNotebook1.pages[pg]);
    for cmpnts := 0 to componentCount - 1 do
      if (components[cmpnts] as TControl).parent = MyPageObj then
        outline1.AddChild(outlineIdx, components[cmpnts].name +

          ' [' + components[cmpnts].ClassName + ']');
  end;
end;
```

As it turns out, there is a slight problem with this code. If a page and its components are added dynamically, this code will not find it. That is because the new component is added to the page's component list and not the form's list. Here is a way around that one:

```
var
  cmpnts, pg: word;
  MyPageObj: TWinControl;
  OutlineIdx: longint;
begin
  for pg := 0 to TabbedNotebook1.pages.count - 1 do begin
    MyPageObj := (TabbedNotebook1.pages.objects[pg]) as TWinControl;
    OutlineIdx := outline1.add(0, TabbedNotebook1.pages[pg]);
    with MyPageObj do
      for cmpnts := 0 to ControlCount - 1 do

        outline1.AddChild(outlineIdx, Controls[cmpnts].name +
          ' [' + Controls[cmpnts].ClassName + ']');
  end;
end;
```

{ This code came from Lloyd's help file! }

How conserve resources with a TTabbedNotebook?

A: The TTabbedNotebook is created with only the page showing actually in memory. As tabs are selected, the pages to be shown are instantiated into memory. The trick is to release the last page from memory so that only one page at a time exists. There are two examples of doing this below.

This version recycles controls by changing the parent property as the pages are turned:

```
procedure TForm1.TabbedNotebook1Change(Sender: TObject; NewTab: Integer;
  var AllowChange: Boolean);
var
  CurrentPage, NewPage: TWinControl;
begin
  if sender is TTabbedNotebook then
    with TTabbedNotebook(sender) do begin
      CurrentPage := TWinControl(pages.objects[PageIndex]);
      LockWindowUpdate(handle);
      NewPage := TWinControl(pages.objects[NewTab]);

      while PresentPage.ControlCount > 0 do
        PresentPage.Controls[0].Parent := NewPage;
      LockWindowUpdate(0);
    end;
end;
```

This version is a bit more elegant and releases he window handle of the control while keeping all the other properties intact. (i.e. You don't have to worry about losing the information contained in that window just because the window went away.)

```
procedure TForm1.TabbedNotebook1Change(Sender: TObject; NewTab: Integer;
  var AllowChange: Boolean);
var
  CurrentPage: TWinControl;
begin
  if sender is TTabbedNotebook then
     with TTabbedNotebook(sender) do begin
      CurrentPage := TWinControl(pages.objects[PageIndex]);
      LockWindowUpdate(handle);
      TWinControl(pages.objects[NewTab]).HandleNeeded;

      LockWindowUpdate(0);
    end;
end;
```

{ This code came from Lloyd's help file! }

Iterate a Table and use outline control

```
procedure TfrmWizDlg.Button3Click(Sender: TObject);
  var
  t: TTable;
  indx, FieldCounter: integer;
begin
t := TTable.create(self);
with t do
begin
  DatabaseName := 'CWBASEII'; {personal alias}
  TableName := 'useddims.db';
  open;
  first; {probably redundant}
  while not eof do
  begin
    indx := outline1.add(0, fields[0].AsString);
    for FieldCounter := 1 to fieldcount -1  do
      outline1.addChild(indx, fields[FieldCounter].AsString);
    next;
  end; {end of the while statement}

  close;
end;  { End of the with statement. }
end; {end of the procedure}
```

Flush the dataBuffer to the table? (i.e. hard write to disk)

A: This will allow you to commit changes to disk without incurring the performance hit of LocalShare. You might write a procedure something like this:

```
{ This will work for TQueries and TTables. }
procedure FlushBuffer(DataSet: TDataSet);
begin
  with DataSet do begin
    UpdateCursorPos;
    check(dbiSaveChanges(Handle));
    CursorPosChanged;
  end;
```

```
end;
```

note: runs with version 2

{ This code came from Lloyd's help file! }

How can I tell if share is loaded from Delphi?

A:

```
function IsShareLoaded: Boolean;
var
  f: file of word;
  data: word;
  IsShareInstalled: Boolean;
begin
  assign(f, 'im_here.not');
  rewrite(f);
  write(f, data);
  asm
    mov IsShareInstalled, true
    mov bx, TFileRec(f).handle
    xor cx, cx
    xor dx, dx
    mov si, 0
    mov di, 2
    mov al, 0
    mov ah, $5C
    int $21
    jc  @@NoError
    dec IsShareInstalled
    @@NoError:
  end; {asm section}

  result := IsShareInstalled;
  close(f);
  erase(f);

end;
```

{ This code came from Lloyd's help file! }

How can I check to see if there is a disk in the "A" drive?

A:

```
var
  ErrorMode: word;
begin
  ErrorMode := SetErrorMode(SEM_FailCriticalErrors);
  try
    if DiskSize(1) = -1 then
      ShowMessage('Drive not ready');
  finally
    SetErrorMode(ErrorMode);
  end;
end;
```

{ This code came from Lloyd's help file! }

What is a Callback function, and how do I create one?

A: A call back function is a function which you write, but is called by some other program or module, such as windows. To create a callback function, you must first declare a function type, the funciton itself, and implement the function. In the interface section:

```
{ In main program interface }

type
  TCallBackFunction = function(s: string): integer;
  CallMe(s: string): integer;
```

And in the Implementation section:

```
{ In main program implementation }

procedure TestCallBack(CallBackFunction: TCallBackFunction); far; external 'Other';
{ Note that 'other' is a Dll containing the procedure TestCallBack }

function CallMe(s: PChar): integer;
begin
  { what ever you need to do }
  CallMe := 1; { What ever you need to return }
end;

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

Note that in 'Other' you would also declare a function type, and use it like this:

```
{ in library Other interface }
type
  TMainFunction = function(s: string): integer;
  TestCallBack(MainFunc: TMainFunction);
{ in library Other implementation }
TestCallBack(MainFunc: TMainFunction);
var
  result: integer;
begin
  result:=MainFunc('test');
end;
```

{ This code came from Lloyd's help file! }

Inc versus x:=x + 1 in assembler test

This is my chance to plug how great Delphi is.

```
x := x + 1;
```

yields the following assembly code:

```
mov  ax,  [bp-02]
inc  ax
mov  [bp-02], ax
```

Now, inc(i) does it this way:

```
inc  word ptr [bp-02]
```

{ This code came from Lloyd's help file! }

Store Data beyond year 2000

Q: If you have the date 1/1/2000 and run the following code: DateToStr(StrToDate('1/1/2000')) the year gets changed from 2000 to 1900. What's the best way to handle this?

A: The value of the typed constant ShortDateTime in WIN.INI determines how DateToStr and StrToDate convert strings. The default is dd/mm/yy - two digit year. I believe setting ShortDateTime := 'dd/mm/yyyy' will solve your problem.

{ This code came from Lloyd's help file! }

Start an exe from Delphi

FindWindow

{This code will launch and close Notepad from a checkbox. FindWindow is the one to use in Win95 as hInstance will always be NIL in Win95. This is so because each program gets its own space.}

```
procedure TForm1.CheckBox1Click(Sender: TObject);
var
  NotepadHandle: hWnd;
begin
  if checkbox1.checked then {start/stop checkbox}
    WinExec('c:\windows\notepad.exe c:\autoexec.bat', sw_ShowNormal)
  else begin
    NotepadHandle := FindWindow('Notepad', nil);
    SendMessage(NotepadHandle, wm_Close, 0, 0);
  end;
end;
```

{ This code came from Lloyd's help file! }

Fill a list (TList) with records

```
type cwObjs = record
       intVal: integer;
       strval: string;
     end;
     PcwObjs = ^cwObjs;

var myList: Tlist;
    myRecPtr: Pcwobjs;


procedure FillList;
var i: integer;
begin
  myList:= TList.create;
  for i:= 1 to 100 do begin
    new(myRecPtr);
    mylist.add(myRecPtr);
    with myRecPtr^ do begin
      intval:=i;
      strval:='kleiner kommunikation';
    end;
  end;
end;

procedure showlist;
var i: integer;
begin
  with frmWizDlg.listbox1 do begin
    clear;
    for i:= 0 to myList.count-1 do begin
      items.add(IntToStr(pcwObjs(myList.items[i])^.intval));
      items.add(pcwObjs(myList.items[i])^.strval);
    end;
  end;
end;


procedure deleteList;
var prompt: string[5];
begin
  if inputQuery('ListDemo','Which one delete',prompt) then begin
  myList.delete(strToInt(prompt)-1);
  showList;
  end else
  exit
end;
```

Release Notes maXbox 4.7.5.20 Jan 2021 mX47

Add 25 Units + 4 Tutorials
1277 unit uPSI_SystemsDiagram.pas Dendron
1278 unit uPSI_qsFoundation.pas Dendron
1279 uPSI_JclStringLists2 JCL
1280 uPSI_cInternetUtils2 FLC
1281 uPSI_cWindows.pas FLC
1282 uPSI_flcSysUtils.pas +TBytes utils
1283 unit uPSI_RotImg.pas DA
1284 uPSI_SimpleImageLoader.pas LAZ
1285 uPSI_HSLUtils.pas LAZ
1286 uPSI_GraphicsMathLibrary.pas EF
1287 unit uPSI_umodels.pas DMath
1288 uPSI_flcStatistics.pas FLC5
1289 uPSI_flcMaths.pas FLC5
1290 uPSI_flcCharSet.pas
1291 uPSI_flcBits32.pas
1292 uPSI_flcTimers.pas
1293 uPSI_cBlaiseParserLexer.pas
1294 uPSI_flcRational.pas
1295 uPSI_flcComplex.pas
1296 unit uPSI_flcMatrix (uPSI_flcVectors.pas)
1297 unit uPSI_flcStringBuilder.pas
1298 unit PJResFile_Routines;
1299 uPSI_flcASCII.pas
1300 uPSI_flcStringPatternMatcher;
1301 unit uPSI_flcUnicodeChar.pas;

Totals of Function Calls: 33282
SHA1: of 4.7.5.20 D82EAD01C58738887661428F94B207DB1D8FAEB5
CRC32: 203C82F0 29.5 MB (31,012,768 bytes
mX4 package maxbox4.zip SHA1:
59e0e6da4bc8a1c4ee4754cf16ae6f26289185bd