Delphi Incomplete reference

Please send me feedback

My email: cdchapman2001@gmail.com

Created and edited by Connor Chapman from 2018 to 2020

For a better reference visit: http://www.delphibasics.co.uk/

Ffixed/ffcurency:

//not this can only be done with a real/float variable

FloattostrF(rNum, ffcurency, 8, 4)

FloattostrF(rNum, Ffixed, 8, 4)

//8 : numbers in front of the comma or fullstop

//4 : decimal places

Date/Time:

ShowMessage('Today = '+DateToStr(Date));
Today = 29/10/2002

Message dialog:

Taken off http://www.delphibasics.co.uk/

```
var
  buttonSelected : Integer;
begin
        buttonSelected := messagedlg('Custom
dialog',mtCustom, [mbYes,mbAll,mbCancel], 0);   // Show a custom dialog
        // Show the button type selected
        if buttonSelected = mrYes    then ShowMessage('Yes pressed');
        if buttonSelected = mrAll    then ShowMessage('All pressed');
         if buttonSelected = mrCancel then ShowMessage('Cancel pressed');
end;
```

//or:

```
if messageDlg('Sure you want to delete recod for ' + sVeg +'?',
mtWarning,[mbOk, mbCancel], 0) = mrOk
```

Passwords:

Edit2.PasswordChar := '•';

Random:

iRandom := RANDOM(10) + 1;

//random(10) will give you a random value from 0 to 9 the plus one will give you 1 to 10

Jpeg:

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ExtCtrls, ComCtrls, **JPEG**;

img.picture.LoadFromFile('image.jpg');

Text files:

Reading from:

```
if FileExists('Names.txt') <> true
        ShowMessage('File does not exist');
        Exit;
end;


AssignFile(fileRead, 'Names.txt');
Reset(fileRead);


while NOT EoF (fileRead)//while it is not end of file
//only use while loops for looking in text files
        ReadLn(fileRead, sLine); //assign sline the current line
        iLength := length(sLine);
        if SLine = ' ' then delete(sLine,1, iLength);
        iPos := POS('#',SLine);
        sName := COPY(sLine, 1, iPos - 1);
        DELETE(sLine, 1, iPos);
        iAge := strtoint(sLine);
end;
```

Writing to:

```
AssignFile(myFile, 'Test.txt');
ReWrite(myFile);

append(myFile);

WriteLn(myFile, sline);

CloseFile(myFile);
```

Sorting:

```
for iLoop := 1 to iCount - 1

        for iLoop2 := iLoop + 1 to iCount

                if arrHeight[iLoop2] < arrHeight[iLoop]

                        sTemp := inttostr(arrHeight[iLoop]);

                        arrHeight[iLoop] := arrHeight[iLoop2];

                        arrHeight[iLoop2] := strtoint(sTemp);

end;end;end;
```

Array:

Array(2d):

```
arrTwoD : Array [1..10, 1..10] of integer;

arrTwoD[iRow][ iCol];
```

String grid:

```
stgDisplay.Cells[iCol, iRow]

stgDisplay.Cells[iCol, iRow]
```

Databases not sql:

Number of records:

```
iCount := adoFruitandveg.RecordCount;
```

Displaying a field:

```
adoFruitandveg.Open;
adofruitandveg.first;
while not adofruitandveg.eof
        redOut.Lines.Add(adofruitandveg['FVName']);
        adofruitandveg.next;
end;
adofruitandveg.Close;
```

Searching for the first instance:

```
adoFruitAndVeg.Open;
if adoFruitAndVeg.Locate('FVname', sVeg, []) = true
        redOut.Lines.Add(adoFruitAndVeg['FVName'] + ' found in position ' +
        inttostr(adoFruitAndVeg['ID']));
end
else
        redOut.Lines.Add('not found in database');
end;
adoFruitandveg.Close;
```

Feld by name

```
adoQryBookings.FieldByName('StartOfBooking').AsDateTime
```

Searching for multiple instances:

```
adoFruitandveg.Open;
adoFruitandveg.First;
while not adoFruitandveg.Eof
        if uppercase(adoFruitandveg['colour']) = uppercase(sCol)
                sOutput := sOutput + adoFruitandveg['FvName'] + ',';
        end;
        adoFruitandveg.Next;
end;
adoFruitandveg.Close;
```

Sorting data:

```
adoFruitandveg.Open;
adofruitandveg.First;
adofruitandveg.Sort := 'Price DESC';
while not adofruitandveg.Eof
        redOut.Lines.Add(floattostrF(adoFruitandveg['Price'], ffCurrency,8,2)
        + #9 + adoFruitandveg['FVName']);
        adoFruitandveg.Next;
end;
adoFruitandveg.Close;
```

Inserting data:

```
sName := inputBox('','Enter name','');
adoFruitandveg.Insert;
adoFruitandveg['FVName'] := sName;
adoFruitandveg.Post;
```

Updating a record:

```
adoFruitandveg.Open;
adoFruitandveg.First;
while not adoFruitandveg.Eof
        adoFruitandveg.edit;
        if uppercase(adoFruitandveg['FVName']) = uppercase(sVeg)
                adoFruitandveg['Colour'] := sCol;
        end;
        adoFruitandveg.Post;
        adoFruitandveg.Next;
end;
adoFruitandveg.Close;
```

Deleting a record:

```
adoFruitandveg.Open;
adoFruitandveg.First;
if AdoFruitandveg.Locate('FVName', sVeg , [])
        if messageDlg('Sure you want to delete recod for ' + sVeg +'?',
        mtWarning,[mbOk, mbCancel], 0) = mrOk
                adoFruitandVeg.Delete;
        end;
end;
adoFruitandveg.Close;
```

Filtering:

adoTable.filter('string');

adotable.filtered;//can't use this in tests

Methods:

Searching for a substring:

Containstext('help me', help) //this returns true or false depending on if a the substring is found or not

String and character Methods:

CHAR	CHAR(65) (= 'A')	Converts an ascii value to a character
ORD	ORD('A') (= 65)	Finds the ascii value of a character
LENGTH	LENTH('Sally') (= 5)	Counts the number of characters in a String
UPPERCASE	UPPERCASE(sally) (= SALLY)	Converts the whole String to uppercase
POS	POS('#', 'Sally#18') (=6)	Finds the position of a substring in a String
COPY	COPY('Frederick', 1, 3) (= 'Fre')	Copies 3 characters from the existing String from index 1
DELETE	DELETE('Fredericko',5,6) (= 'Fred')	Deletes 6 characters from the existing String from index 5
TRIM	TRIM (' spaces ') (= 'spaces')	Cuts off white space before and after the actual characters in the String

Mathematical Methods:

DIV	10 DIV 5 (= 2)	Divide integers to get an integer answer
MOD	10 MOD 5 (= 0)	Remainder after dividing
INC	INC(2) (= 3)	Adds 1 onto the existing number
DEC	DEC(2) (= 1)	Subtracts 1 from the existing number
TRUNC	TRUNC(8.7) (= 8)	Cuts off the decimal part and leaves the whole number
ROUND	ROUND(8.7) (= 9)	Rounds off the existing number
FRAC	FRAC(8.7) (= 7)	Cuts off the whole number and returns the decimal part
SQR	SQR(4) (= 16)	Squares the number
SQRT	SQRT(16) (= 4)	Finds the square root of the number
ABS	ABS (-4) (= 4)	Ensures that the number is positive in all cases
POWER	POWER(2,3) (= 8)	Raises the 1st number to the power of the 2nd number

Procedures:

Declaring:

```
procedure blablabla(sIn: string; var sOut: string);
    sOut := sIn;
end;
```

Calling:

```
blablabla (sIn, sOut);
```

Functions:

Declaring:

```
function getString(sAnyString : string) : String;
var
        sTheoreticalString : string;
begin
        sTheoreticalString := sAnyString
        Result := sTheoreticalString;
end;
```

calling:

```
sSRealtr := getString(sAnyString);
```

Dynamic objects:

```
btnTemp: TButton; //step 1 - Declare your global object

procedure TForm1.StaticButton1Click(Sender: TObject);

        //step 2 - create your dynamic button

        btnTemp := TButton.Create(Form1);

        // Step 3 - Set properties of your object

        btnTemp.Caption := 'Say hello';

        with btnTemp do

                Height := StaticButton1.Height; // did this so that the objects
                could be placed in relation to the other

                Width := StaticButton1.Width;

                Top := StaticButton1.Top + StaticButton1.Height;

                Left := StaticButton1.Left + StaticButton1.Width;

                Parent := Form1;

        end;

        //Step 5 - assign your methods to your object events

        btnTemp.OnClick := btnTempwhenClicked;

end;

//Step 4 - create procedure for your methods

procedure TForm1.btnTempwhenClicked(Sender: TObject);

begin

        showmessage('Hello!');

end;
```

More advanced:

```
for iLoop := 1 to 9 do
        arrBtnNumbers[iLoop] := TButton.Create(Form1);
        with arrBtnNumbers[iLoop] do
                Parent := Form1;
                Caption := '+' + inttostr(iLoop);
                width := 75;
                height := 25;
                left := 8;
                if iLoop = 1 then
                        top := StaticBtnReset.height + StaticBtnReset.top + 1
                else
                        top := arrBtnNumbers[iLoop - 1].height +
                        arrBtnNumbers[iLoop - 1].top + 1;
                onClick := NumberClick;
        end;
  end;
procedure TForm1.NumberClick(Sender: TObject);
begin
        iSum := iSum + strtoint((Sender as TButton).Caption[2]);
        lblSum.Caption := inttostr(iSum);
end;
```

Object orientated programming:

Separate unit:

```
TMatricDance = class // data type
private
        fBoy: string;
        fGirl: string;
public
        constructor create; overload; // allways create
        constructor create(sB, sG: string);overload;
        //mutators
        procedure setBoyName(sB: string); // always set
        procedure setGirlName(sG: string);
        // accessors
        function getBoyName: string;
        function getGirlName: string;// always get
        // auxillary
        function toString: string;
```

primary unit:

```
var
  Form1: TForm1;
  couple: TMatricDance;
```

constructing in primary unit:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  sBoy: string;
  sGirl: string;
begin
        sBoy := inputbox('', 'Boy''s name', '');
         sGirl := inputbox('', 'Girl''s name', '');
        couple := TMatricDance.create(sBoy, sGirl);
        RichEdit1.lines.add(couple2.toString);
end;
```

Mutators:

```
procedure TMatricDance.setBoyName(sB: string);
begin
  fBoy := sB;
end;
```

Accessors:

```
function TMatricDance.getBoyName: string;
begin
  result := fBoy;
end;
```

Databases sql: refer to online resources for more sql stuff

```
SELECT * FROM Customers;
SELECT [CustomerID] FROM Customers;
SELECT DISTINCT[CustomerID] FROM Customers;
SELECT TOP 5 CustomerName FROM Customers;
SELECT CustomerName, City FROM Customers WHERE City = London;
SELECT * FROM Customers ORDER BY CustomerID DESC;
SELECT COUNT([*]) AS NumRecords FROM Customers;
SELECT SupplierID, SUM([Price]) AS TotPoP FROM Products GROUP BY
SupplierID;
SELECT SupplierID, AVG([Price]) AS AvgPoP FROM Products GROUP BY
SupplierID;
SELECT SupplierID, MIN([Price]) AS TotPoP FROM Products GROUP BY
SupplierID;
SELECT SupplierID, FIRST([Price]) AS AvgPoP FROM Products GROUP BY
SupplierID;
SELECT SupplierID, AVG([Price]) AS AvgPoP FROM Products GROUP BY
SupplierID HAVING NOT SupplierID = 1;
INSERT INTO Products (ProductID, ProductName, SupplierID, CategoryID, Unit,
Price) VALUES (100, 'Gumbo Mix', 1, 2, '30 boxes', 22);
UPDATE Products SET SupplierID = 101 WHERE SupplierID = 1;
DELETE FROM Products WHERE SupplierID = 101;
SELECT Customers.CustomerID, Orders.OrderID FROM Customers,
Orders WHERE Orders. CustomerID = Orders. CustomerID;
SELECT SupplierID FROM Products Union SELECT CustomerID FROM
Customers;
```