XPM | Coding Standards

This page was designed to remind us of the standards we should use while creating or maintaining our codes in Delphi.

UNITS

- The name of the unit should be in mixed upper and lowercase.

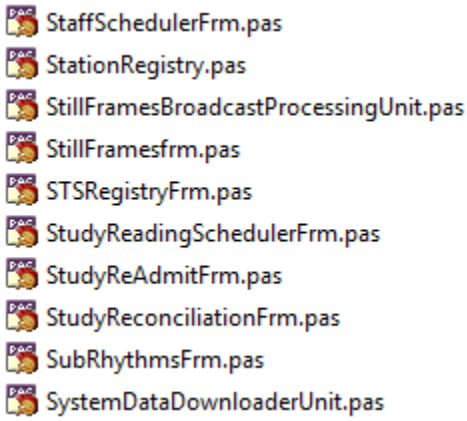
File unit example:

Good:

MyUnit.pas

Bad:

myUnit.PAS



CLASSES

- NAMES
  - Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser;
  - Avoid words like Manager, Processor, Data, or Info in the name of a class;
  - A class name should not be a verb.

- DECLARATION
  - Each field shall be declared with a separate type on a separate line;
  - Type names for classes will be meaningful to the purpose of the class. The type name must have the T prefix to annotate it as a type;
  - Instance names for classes should match the type name of the class without the T prefix.

Example:

```
TNewClass = class(TObject)
private
  FField1: Integer;
  FField2: Integer;
  FField3: String;
```

Declaration Example:

```
type
  TCustomer = class(TObject)
```

Instance Example:

```
var
  Customer: TCustomer;
```

METHODS

- Methods should have verb or verb phrase names like “postPayment”, “deletePage”, or “save”;
- Do not use underscores to separate words;
- Method names should be imperative verbs or verb phrases.

Class example:

```
public class Version {
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

CONSTANTS

- They are always in upper case.
- Words are separated by the underscore character (_).

Declaration Example:

Good:

```
const
  WM_MY_MESSAGE = WM_USER + 0;
  WM_ANO_MESSAGE = WM_USER + 1;
```

Bad:

```
const
  WMMYMESSAGE = WMUSER + 0;
  Wm_Ano_Message = Wm_User + 1;
```

VARIABLES

- Use meaningful names to the variable's purpose.
- Boolean variable names must be descriptive enough so that their meanings of True and False values will be clear.
- Variables should be grouped by its type.

Example:

Good:

```
Code: String
DateEmission: TDateTime;
Number: Integer;
```

Bad:

```
Code: String
DateEmission: TDateTime;
ntNumber: Integer;
```

VARIABLES - ENUMERATED TYPES

- Enumerated Types example

Good:

```
TSongType = (stRock, stClassical, stCountry);
type
  TSongKind = (stUndefined, stRock, stJazz, stClassical, stCountry);
```

Bad:

```
TSongType = (stRock, stClassical, stCountry);
type
    TSongTypes = (RockSong, JazzSong, ClassicalSong, CountrySong);
```

VARIABLES - ARRAY TYPES

- Names for array types must be meaningful to the purpose of the array;
- The type name must be prefixed with a T character;
- If a pointer to the array type is declared, it must be prefixed with the character P and declared immediately prior to the type declaration.

Example:

```
type
  PCycleArray = ^TCycleArray;
  TCycleArray = array[1..100] of Integer;
```

VARIABLES - RECORD TYPES

- A record type shall be given a name meaningful to its purpose;
- The type declaration must be prefixed with the character T;
- If a pointer to the record type is declared, it must be prefixed with the character P and declared immediately prior to the type declaration.

Example:

```
type
  PEmployee = ^TEmployee;

  TEmployee = record
    Name: string;
    Rate: Double;
  end;
```

COMMENTS

- One of the more common motivations for writing comments is bad code;
- We write a module and we know it is confusing and disorganized;
- It is better to comment that? No! You'd better clean it!
- Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments;
- Rather then spend your time writing the comments that explain the mess you've made, spend it cleaning that mess.

Line example:

Good:

```
i := i + 1; // Add one to i
```

Bad:

```
// Add one to i
i := i + 1;
```

Block example:

Good:

```
{
 TMyObject.MyMethod
  This routine allows you to execute code.
}
```

Bad:

```
{  TMyObject.MyMethod

  This routine allows you to execute code.}
```

MARGINS

- The source shall not exceed the margin with the exception to finish a word;
- Use operators like (AND, OR) at the end line, never at begin line;
- Start the second line and the subsequent aligned with the start of the first sentence;

- When the begin of a block is already very close to the edge is a strong indication that the method should be broken into two or more smaller methods. Check the ability to refactor the method.
- Try to use a maximum of 140 characters per line.

- WRONG

```
3753      else
3754      if (VarIsNull(edtLabHGBSI.EditValue) OR (edtLabHGBSI.Text = '')) OR (VarIsNull(edtLabHGBSImmol.EditValue) OR (edtLabHGBSIm
3755      begin
3756        GL_CaseData.tblLabValues.Edit;
```

- CORRECT

```
3753      else
3754      if (VarIsNull(edtLabHGBSI.EditValue) OR (edtLabHGBSI.Text = '')) OR (VarIsNull(edtLabHGBSImmol.EditValue) OR
3755        (edtLabHGBSImmol.Text = '')) then
3756      begin
3757        GL_CaseData.tblLabValues.Edit;
```

INDENTATION

- To make this hierarchy of scopes visible, we indent the lines of source code in proportion to their position in the hierarchy;
- Statements at the level of the file, such as most class declarations, are not indented at all;
- Methods within a class are indented one level to the right of the class;
- Implementations of those methods are implemented at one level to the right of the method declaration;
- Block implementations are implemented one level to the right of their containing block, and so on.
- Indenting will be two spaces per level.

Example:

Good:

```
if SomeValeu = 1 then
begin
  for x := 0 to 10 do
  begin
    ToDoSomething;
  end;
end
else
begin
  ToDoOhertThing;
end;
```

Bad:

```
if SomeValeu = 1 then begin
for x := 0 to 10 do
begin
  DoSomething;
end;
end else
begin
  DoOhertThing;
end;
```

- Character Spacing
  - Spaces must be used around all operators.
  - Space after all commas.
  - No space between a variable declaration and the semicolon.
  - Spaces around the = in type and constant declarations.
  - There must be no white space between parentheses and the surrounding characters in procedure and function calls.
  - There shall never be white space between an open parenthesis and the next character.

Examples:

Good:

```
Type
  TCustomer = class(TObject);

implementation

var
  S: string;

procedure DoSomething(Parameter);

if (I = 42) then

J := Max(K * 12 / 3) - 5;

CallProc(Parameter);
```

Bad:

```
Type
   TCustomer=class(TObject);

implementation

var
J : integer;

procedure DoSomething ( Parameter );

if (I=42)then

J := Max(K*12/3)-5;

CallProc( Parameter );
```

Variable declaration

- Variable types shouldn't be aligned using TAB characters.

Good:

```
var NumberTime: Integer;
var UserName: String;
```

Bad:

```
var NumberTime:      Integer;
var UserName:        String;
```

Case condition

- All separate parts of the case statement have to be indented;
- All separate parts of the case statement have to have a line between each other.
- All condition statements shall be written in (begin/end) blocks;
- The else clause aligns with the case statement.

Example Case:

```
case ScrollCode of
  SB_LINEUP, SB_LINEDOWN:
  begin
    Incr := FIncrement div FLineDiv;
    FinalIncr := FIncrement mod FLineDiv;
    Count := FLineDiv;
  end;

  SB_PAGEUP, SB_PAGEDOWN:
  begin
    Incr := FPageIncrement;
    FinalIncr := Incr mod FPageDiv;
    Incr := Incr div FPageDiv;
    Count := FPageDiv;
  end;

else
  Count := 0;
  Incr := 0;
  FinalIncr := 0;
end;
```

TRY / FINALLY | TRY / EXCEPT

- Where possible, each allocation will be matched with a (try-finally) construct;
- Use (try-except) only when you want to perform some task when an exception is raised.

Try finally example:

```
SomeClass1 := TSomeClass.Create;
If not Assigned(SomeClass1) then
  Exit;
try
  SomeClass2 := TSomeClass.Create;
  If not Assigned(SomeClass1) then
    Exit;
  try
    { do some code }
  finally
    SomeClass2.Free;
  end;
finally
  SomeClass1.Free;
end;
```

Try except example:

```
try
  Calculate;
except
  on EZeroDivide do HandleZeroDivide;
  on EOverflow do HandleOverflow;
  on EMathError do
  begin
    HandleMathError;
    raise;
  end;
else
  HandleAllOthers;
end;
```

SQL CONCATENATION

- Long commands must be broken down into multiple lines using the + operator;
- Align the keywords one below the other;
- WHERE conditions must be aligned one below the other.

Concatenation example:

```
xQuery.SQL.Text:= 'SELECT MAX(VD_INDEX)' +
                    ' FROM VITALSIGNS' +
                    ' WHERE FACILITY_ID=' + QuotedStr(Facility_ID) +
                      ' AND STATION_ID=' + QuotedStr(Station_ID) +
                      ' AND CASE_NO=' + IntToStr(Case_Number) +
                      ' AND VD_INDEX BETWEEN ' + GetWorkStationIDNumber + '0000' +
                      ' AND ' + GetWorkStationIDNumber + '9999' +
                    ' ORDER BY';
```