ICS & MidWare coding style

THIS IS A WORK IN PROGRESS !

1. Keywords
   All Delphi keywords are written in lower case except data types.

```
function Demo(const Value : String): String;
begin
    Result := 'This is my demo';
end;
```

2. Data types
   Built-in data types are written in lower case except the first letter. See example above. All new data types are created using concatenated words in lower case with each first letter in uppercase with preceding letter ‘T’. If data type is a pointer to something then first letter ‘T’ is replaced by ‘P’.

```
TMyOwnDataRecord = record
    Street : String;
    Value  : Integer;
end;
PMyOwnDataRecord = ^TMyOwnDataRecord ;
```

3. Variable and function identifiers
   Variables and functions identifiers are written using concatenated words in lower case with each first letter in uppercase. The names shall be chosen to be as meaningful as possible. Examples :

```
procedure PrintErrors;
function  ReadLine     : String;
function  GetCharacter : Char;
var
    MaxCount : Integer;
```

   Global variables are prefixed with uppercase letter ‘G’:

```
var
    GClassCritSect : TRTLCriticalSection;
```

4. Constant identifiers
   Constant identifiers shall be written in all upercase, with underline character to split the identifier in more readable words.

```
const
    IP_MULTICAST_TTL  = 3;
    IP_MULTICAST_LOOP = 4;
    IP_ADD_MEMBERSHIP = 5;
```

   Typed constant which are in fact initialized variables follow variable indentifier rule. Old Delphi compiler didn’t liked intialized variables, so they where declared with the const keyword and needed the compiler option {$J+} to allow typed constant to be modified.

```
const
```

```
          MyVariable : Integer = 123;
```

When a typed constant is really a global variable, it must be prefixed with uppercase letter 'G'.

Constants which are much like enul can also follow the same rule (see below).

```
      const
          httperrNoError  = 0;
          httperrBusy     = 1;
          httperrNoData   = 2;
          httperrAborted  = 3;
```

5. Enum identifiers
   Enum identifiers will begin with a prefix in lower case being 2 to 4 letters of the words describing the enum datatype. After the prefix, the identifier follow the rules for variables.

```
      type
         THttpEncoding = (encUUEncode, encBase64, encMime);
```

6. Properties
   Properties names follow variable names convention. If the property is mapped to a variable, the variable name is the same as the property name with uppercase 'F' as prefix. If property is mapped with accessor functions, the their names begin with « Get » or « Set ».

```
      private
          FValue : Integer;
          function  GetLine : String;
          procedure SetLine(const Value : String);
      published
          property Value : Integer read FValue  write FValue;
          property Line  : String  read GetLine write SetLine;
      end;
```

7. Events
   Same as properties except the « On » prefix is added.

```
      private
          FOnModify : TNotifyEvent;
      published
          property OnModify : TNotifyEvent read  FOnModify
                                           write FOnModify;
      end;
```

8. Indentations, tabs and spaces
   Indentation is by 4 spaces. Do not use tab character, always insert spaces characters so that the code is displayed the same way whatever the tab settings are.

9. Alignments
   Align everything as much, using spaces. Align in variables, records and classes declarations, align in assignment. See how alignments are done in the properties example above : there is an extra space between « function » and « GetLine » so that « GetLine » is aligned with « SetLine » on the next line. See also how the colon are aligned in the properties section. Same alignment for « read » and « write » keywords.
   Argument have their colon aligned :

```
function TCustomSyncWSocket.Synchronize(
    Proc         : TWSocketSyncNextProc;
    var DoneFlag : Boolean) : Integer;
begin
    DoneFlag := FALSE;
    if Assigned(Proc) then
        Proc;
    Result := WaitUntilReady(DoneFlag);
end;
```

10. Comments

Since most ICS components are for all Delphi version, including Delphi 1, only old style comment are allowed. Midware and some ICS component are for later Delphi so new comment style are allowed.

Whenever possible, alignement have to be respected not only at the left but also at the right margin.

Comments describing function behaviour has to be put before the function itself. Comments within the function body is aligne with code and is placed before the code if the comment apply to several lines, or is too long to fit at the right of the code.

Each function is preceded by a comment line made of 38 asterisk and spaces. At the end of the function there are exactly two blank laines before the next function.

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{ This comment describe the function behaviour. You should include anythin  }
{ useful to explain what the function does, and what it returns.            }
function ThisIsMyFunction(
    Value        : Integer;              { This describe the argument       }
    const Street : String;               { Yet another argument description }
    LastArg      : Byte) : Integer;      { Here the last argument described }
var
    I      : Integer;
    Buffer : String;
begin
    { This comment applies to the next fex lines.                           }
    SetLength(Buffer, 20);               { Allocate string space            }
    for I := 1 to Length(Buffer) do
        Buffer[I] := Char(Ord('A') + I); { Fill buffer with some data       }

    { New code block with his own comment.                                  }
    DoSomething;                         { Another comment for this line    }
    Result := 3;
end;


{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
```

Note in the above example that all other rules are also respected. Take special care to alignement of everything. Of course it is not always possible to align everything because the lack of space. Do your best to preserve as much alignments as possible, breaking the less rules as possible.

11. Begin/End keywords

Begin is always at the end of line, end keyword is always aligned with the if, while, for which has the matching begin.

```
if Value = DefaultValue then begin
```

```
        DoSomething;
        DoAnotherThing;
    end
    else begin
        I     := 20;
        Value := I;
    end;

    for I := 1 to 10 do begin
        DoSomething;
        DoAnotherThing;
    end;

    while I < 10 do begin
        DoSomething;
        Inc(I);
    end;
```

12. Comma, semi-colon, parenthesis, bracket
    Function arguments are separated by comma when calling the function. Put a space after the comma but none before. Same with the semi-colon in the function declaration.
    The semi-colon which ends a code line has no preceding space.
    An open parenthesis is never preceded nor followed by a space. Same for brecket in array indexing.

13. Conditional expressions
    Conditional expression do not use leading and ending parenthesis. When the expression is made of several subexpressions, parenthesis are used around each part. A space is used before and after each operator. No space after opening parenthesis nor before closing parenthesis:

```
    while (I <= Length(Str)) and (Str[I] = ' ') do
        Inc(I);
```

    When a condition is too long to fit on a single line, it is split on several lines. The boolean operator is left at the end of the line and the opening parenthesis are aligned:

```
    if (FAuthNTLMState =  ntlmMsg3) and
       (FStatusCode <> 401) and (FStatusCode <> 407) then
        FAuthNTLMState := ntlmDone
```

14. Line break
    Always use at most 80 characters lines. You are helped with the margin Delphi editor can display for you. Turn on margin display and set it to 80 characters (this is the default anyway).

    Long lines will be continuated at the next line, preserving alignment as much as possible. When breaking a line in two parts, adjust the left margin so that it makes a “logical” alignment with the line above.

    When a function has too much arguments to fit on a single line, the opening parenthesis is where to break the line. Then each argument is listed one by line. Always align inside argument declaration.

```
function ThisIsMyFunction(
    Value        : Integer;              { This describe the argument       }
    const Street : String;               { Yet another argument description }
    LastArg      : Byte) : Integer;      { Here the last argument described }
```

15. <u>Conditional compilation</u>
    Conditional compilation directives start at the left margin when possible.

```
begin
{$IFNDEF NO_DEBUG_LOG}
    if CheckLogOptions(loProtSpecInfo)
        DebugLog(loProtSpecInfo, 'RequestDone');
{$ENDIF}
    if Assigned(FOnRequestDone) then
        FOnRequestDone(Self, FRequestType, FRequestDoneError);
end;
```

    Identation is not affected by the conditional compilation.

    Sometimes conditional compilation apply to only part of a statement. In that case, it is allowed to put the directives in line with the statement:

```
procedure MyOwnProc); {$IFNDEF BCB} virtual; {$ENDIF}
```

16. <u>try/finally and try/except</u>
    Try, finally and related end are all aligned. Indent by four spaces between try and finally and between finally and end. Same for try/except.

```
EnterCriticalSection(GWSockCritSect);
try
    if WSocketGCount <= 0 then
        WSocketUnregisterClass;
finally
    LeaveCriticalSection(GWSockCritSect);
end;



try
    DoSomething;
except
    on E:Exception do begin
        Display(E.ClassName + ': ' + E.Message);
        DoSomethingElse;
    end;
end;
```