**Procedural Type & Method Pointer in Delphi**

**Procedural Type :**

Procedural type is one of user defined data type that allows you to consider procedures and functions as values that can be assigned to variables or passed to other procedures and functions. It is mostly used when we call DLL methods in Dynamic call, to declare custom Events for components.

The following example demonstrates usage of a procedural type.

Suppose you define a function called Calc that takes two integer parameters and returns an integer:

function Calc(X,Y: Integer): Integer;

You can assign the Calc function to a variable F if we declare F as function type.

var   
   F: function(X,Y: Integer): Integer;  
.....

   F := Calc;

**How to declare? :**

If you take any procedure or function heading and remove the identifier after the word procedure or function, then what is left is the right part of a procedural type declaration.

You can use such type names directly in variable declarations (as in the previous example) or to declare new types:

type

   TIntegerFunction = function: Integer;

   TProcedure = procedure;

   TStrProc = procedure(const S: string);

   TMathFunc = function(X: Double): Double;

var

   F: TIntegerFunction; // F is a parameterless function that returns an integer

   Proc: TProcedure;    // Proc is a parameterless procedure

   SP: TStrProc;        // SP is a procedure that takes a string parameter

   M: TMathFunc;        // M is a function that takes a Double (real)

   procedure FuncProc(P: TIntegerFunction);

// FuncProc is a procedure whose only parameter is a parameter less integer-valued function

**Method Pointers :**

Method Pointers reference a method of an instance object. All the variables shown in the previous example are all method pointers – which points to the address of a procedure or function. If you want to reference a method of an instance of a Classes then you need to add the word **“of object”** to the procedural type name.

For example:

type

   TMethod      = procedure of object;

   TNotifyEvent = procedure(Sender: TObject) of object;

These types of method pointers represent a pair of pointers; the first stores the address of a method, and the second stores a reference to the object of the class the method belongs to.

**Assigning a procedure to a variable :**

We can assign two procedural types if they are compatible. Two procedural types are compatible if they have:

1. The same calling convention,
2. The same return value (or no return value),
3. The same number of parameters, with identically typed parameters in corresponding positions. (Parameter names do not matter.)

Example :

type

   TNotifyEvent = procedure(Sender: TObject) of object;

   TMainForm = class(TForm)

      procedure ButtonClick(Sender: TObject);

    ...

end;

var

   MainForm: TMainForm;

   OnClick: TNotifyEvent

We could make the following assignment:

   OnClick := MainForm.ButtonClick;

**Note\*.** The value nil can be assigned to any procedural type.

**Procedural Types in Statements and Expressions**

var

   F: function(X: Integer): Integer;

   I: Integer;

   function SomeFunction(X: Integer): Integer;

     ...

   F := SomeFunction;  // assign SomeFunction to F

   I := F(4);  // call

The first statement assigns a procedural value to F. The second statement copies that value to another variable. The third statement makes a call to the referenced function and assigns the result to I. Because I is an integer variable, not a procedural one, the last assignment actually calls the function (which returns an integer).

In some situations it is less clear how a procedural variable should be interpreted. Consider the statement:

  if F = MyFunction then ...;

In this case, the occurrence of F results in a function call; the compiler calls the function pointed to by F, then calls the function MyFunction, then compares the results. The rule is that whenever a procedural variable occurs within an expression, it represents a call to the referenced procedure or function.

To compare the procedural value of F with MyFunction, use:

  if @F = @MyFunction then ...;