**Version History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Ver. No.*** | ***Authors*** | ***Date*** | ***Reviewers*** | ***Review Date*** | ***Release Date*** |
| 1.0 | Application Development Team | 27-Aug-2018 | QMF | 31-Aug-2018 | 03-Sep-2018 |
| 2.0 | Application Development Team | 10-Dec-2019 | QMF | 13-Dec-2019 | 16-Dec-2019 |
| 3.0 | Application Development Team | 02-Nov-2020 | QMF | 06-Nov-2020 | 10-Nov-2020 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Change History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Ver. No.*** | ***Section*** | ***Date*** | ***Change Information*** | ***RFC No.*** |
| 1.0 | All | 03-Sep-2018 | New Release | - |
| 2.0 | All | 03-Sep-2018 | New Release | - |
| 3.0 | All | 02-Nov-2020 | Annual Review | - |
|  |  |  |  |  |
|  |  |  |  |  |

# STANDARDS FOR CODING IN VISUAL BASIC

1. **Objective**

The objective of this document is to define coding standards for programs developed in ‘Visual Basic ’.

1. **Scope**

This standard applies to programs developed in ‘Visual Basic ’.

1. **Document Organization**

This document is organized as follows:

* Source Code
* Program Organisation and Style
* Data Types
* Layout Guidelines
* Procedures and Functions
* Error Handling
* Naming Conventions
* Comments and Headers
* Menus
* Custom Controls
* Common dialogs
* Validation and Testing
* Help
* Features of Visual Basic 5.0
* Features of Visual Basic 6.0

1. **Source Code**

## Projects

* Visual Basic source code is made up of forms (.FRM files) , modules (.BAS files) , Class Modules (.cls) , Active X EXE/DLL , Active X Controls (.OCX , .CTL) and Active X Documents (.DOB). The configuration management of Visual Basic source code should involve the control of forms, modules, class modules, Active X components and Active X documents
* Forms should contain as little code as possible as this facilitates reuse and reduces Form load time

## Windows API calls

* The 32-bit versions of Windows provide Dynamic Link Libraries (DLLs) in User32.dll, GDI32.dll & Kernel32.dll. These are libraries of procedure external to the application, but can be called from the application. These procedures are referred to as the Windows Application Programming Interface (API). They can be called from any Visual Basic application. Windows API calls should be used
* To accomplish tasks that Visual Basic can not perform
* To enhance performance
* DLLs execute much faster than VB code. If you have a procedure that requires maximum performance, you can write a DLL procedure, and then invoke the DLL from Visual basic
* To have greater modularity and faster application maintenance.
* DLLs get updated independently of the application i.e. they can be updated without recompiling the application that invoked the DLL.
* vbNullstring constant should be used to pass NULL to the DLL

1. **Program Organization and Style**

## Use of VB Constants

* Visual Basic constants should be used as far as possible for e.g. In case of a CheckBox control use Project1.check1.value=vbChecked instead of project1.check1.value=1

## Yielding Control to Windows

* When time-intensive calculations are in progress within a VB application, control should be yielded to Windows using the DoEvents instruction - if appropriate for the application

## Assignment to Variables

* The Let command should not be used when assigning values to variables

## Use of "Me"

* "Me” should not be used to refer to the currently active form; forms should always be explicitly referred to by name. "Me" may be used to refer to nonactive forms, in which events are occurring, E.g. to refer to a timer on a nonactive form

## Branching

* All branching should be conditional
* The ‘Goto’ command should only be used for Error handling (with OnError)

## Concatenation

* The & operator should always be used in preference to the + operator when a concatenation is required

1. **Data Types**

* Option Explicit : Should be used always
* If a variable will always store data of a particular type, the variable should be explicitly declared. This makes data handling more efficient
* E.g. Dim szName As String is better than :- Dim szName which defaults to being a variant
* All variables should be declared before use. The 'Require Variable Declaration' option must therefore be set to 'Yes' in the Environment dialog box to ensure that variables have been declared and to reduce the number of bugs caused by typing mistakes

1. **Layout Guidelines**

## Code Layout

* Code formatting must reflect the logic structure and nesting of the code, while conserving screen space
* Tab-based indentation should be used for blocks. The tab spacing should be four spaces. This can be set using the Environment dialog box
* If Then statements should be structured as shown in the following example,

if <condition> Then

Statement1

Statement2

End If

If..Then blocks should always be terminated with End If, even if only one statement is being conditionally executed.

Long lines of code should be broken where possible, for example,

sQuery = "SELECT COL1, COL2, COL3, ................., COL 8 FROM TABLE 1;"

becomes :-

sQuery = "SELECT COL1, COL2, COL3, COL4, COL5, COL6,"

sQuery = sQuery & " COL7, COL 8 FROM TABLE 1;"

* A single line of code should have a maximum of 72 characters only

## Screen Layout

* + 1. ***System without MDI Concept***
  + Toolbar should always be on the top of the respective form. All forms are MODAL form
    1. ***System with MDI Concept***
  + Toolbar should be on the MDI form instead of on all the forms
  + All forms should be MDI child form
  + No MODAL form should be there in the system except for? About Form?
    1. ***Form (D - Design Time, R- Run Time) Settings***
  + Control Box should be ON. (D)
  + Maximize Button Should be OFF. (D)
  + Border type should be Fixed Double. (D)
  + Caption property should be same as name of the MENU.(D-R)
  + First center the form and then show on the screen (R)
  + Always align objects to the Grid of form. (D)
  + Cursor movement should be logical. (D)
  + All Single line Controls (e.g. Text Box, Combo Box, Display Label, etc.) must have same height. (D)
  + If any other form is activating from activate form then child form should be smaller than the parent form. If it is not possible then give cascading effect. (R)
    1. ***Label***
  + Use Control array for Labels (indicating entry field on the right)
  + Align property should be LEFT ALIGN
  + Background property of LABEL should be transparent
  + Border Style should be 1 for labels, which are used for display purpose only
  + Labels should have hot keys for accessing them through keyboard
    1. ***Command / Group Command Button***
  + For Toolbar use Group Command Button and set Width = 345 and Height = 330
  + Bevel width should be 1 for Group Command Button and Command Button
  + Outline Property should be none
  + Rounded Corner Property Should be FALSE
    1. ***Fonts***
  + Font size should be kept 8.25 i.e. default
  + Font name should be MS-SansSerif i.e. default
  + For 3-D effects of fonts Use LIGHT INSET
    1. ***Tip for Toolbar***
  + Tip should be displayed in Black on Light Yellow color
  + Text appearing in the tip should be assigned to the tag property of the Button
    1. ***Grid Handling***
  + Insertion and Deletion of rows should be available. Preferably on the Right Mouse Button
  + If the data is to be added or edited, it is preferable to do so using textboxes rather than grid
    1. ***Colours of the controls***
  + Use LIGHT GRAY color for form's background
  + Use default color for all other objects
  + Use LIGHT GRAY color for disabled fields. In grid column change color if that column is disable or for display purpose or only for selection

1. **Procedures and Functions**

* Descriptive Variable and Procedure Names
* The body of a variable or procedure name should use mixed case and should be as long as necessary to describe its purpose. In addition, function names should begin with a verb, such as InitNameArray or CloseDialog
* For frequently used or long terms, standard abbreviations are recommended to help keep name lengths reasonable. In general, variable names greater than 32 characters can be difficult to read on VGA displays
* When using abbreviations, make sure they are consistent throughout the entire application. For example, randomly switching between Cnt and Count within a project will lead to unnecessary confusion
* If there are two procedures having the same name in both the modules then the function call needs to be prefixed with the module name every time
* You cannot pass variables of user-defined data types by value. Such variables always need to be passed by reference
* If you specify an optional argument then all subsequent arguments in the argument list need to be declared as optional

1. **Error Handling**

* Centralized Error Handling
* There will be a standard error routine for each project. This routine will be public to the project and will reside in a module file. This error handling routine will display messages as per the error generated. The errors for which specific messages are to be displayed and the errors for which the message returned by VB should be displayed will be application specific and hence the error routine will vary for each project
* At any point in the program, when error trapping is done using the On Error. GoTo, on trapping the error, the program will call the common error routine, with the parameter as the error code

1. **Naming Conventions**

## Naming Constants

* Constants should be in upper case with underscores between words (E.g. MAX\_LINES)
* It may also be useful to add the datatype and scope (see below)
* E.g. Global Const gnMAX\_LINES = 30

## Control Naming Guidelines

* You can use a prefix to describe the class, followed by a descriptive name for the control. Using this naming convention makes the code more self-descriptive and alphabetically groups similar objects in the Object list box. For example, you might name a Check Box control like this: chkReadOnly
* The names you give to forms and controls:
* must begin with a letter
* must contain only letters, numbers, and the underscore character (\_); punctuation characters and spaces are not allowed
* must be no longer than 40 characters
* Suggested Prefixes for Controls

|  |  |  |
| --- | --- | --- |
| **Control Type** | **Prefix** | **Example** |
| 3D Panel | pnl | pnlGroup |
| Animated button | ani | aniMailBox |
| Check box | chk | chkReadOnly |
| Combo box, |  |  |
| drop-down list box | cbo | cboEnglish |
| Command button | cmd | cmdExit |
| Common dialog | dlg | dlgFileOpen |
| Communications | com | comFax |
| Control | ctr | ctrCurrent |
| Data control | dat | datBiblio |
| Data-bound combo box | dbcbo | dbcboLanguage |
| Data-bound grid | dbgrd | dbgrdQueryResult |
| Data-bound list box | dblst | dblstJobType |
| Directory list box | dir | dirSource |
| Drive list box | drv | drvTarget |
| File list box | fil | filSource |
| Form | frm | frmEntry |
| Frame | fra | fraLanguage |
| Gauge | gau | gauStatus |
| Graph | gra | graRevenue |
| Grid | grd | grdPrices |
| Horizontal scroll bar | hsb | hsbVolume |
| Image | img | imgIcon |
| Key status | key | keyCaps |
| Label | lbl | lblHelpMessage |
| Line | lin | linVertical |
| List box | lst | lstPolicyCodes |
| MAPI message | mpm | mpmSentMessage |
| MAPI session | mps | mpsSession |
| Masked Edit Box | med | meddate |
| MCI | mci | mciVideo |
| MDI child form | mdi | mdiNote |
| Menu | mnu | mnuFileOpen |
| MS Flex grid | msg | msgClients |
| MS Tab | mst | mstFirst |
| OLE | ole | oleWorksheet |
| Outline | out | outOrgChart |
| Pen BEdit | bed | bedFirstName |
| Pen HEdit | hed | hedSignature |
| Pen ink | ink | inkMap |
| Picture | pic | picVGA |
| Picture clip | clp | clpToolbar |
| Report | rpt | rptQtr1Earnings |
| Shape | shp | shpCircle |
| Spin | spn | spnPages |
| Text box | txt | txtLastName |
| Timer | tmr | tmrAlarm |
| UpDown | upd | updDirection |
| Vertical scroll bar | vsb | vsbRate |
| Slider | sld | sldScale |
| ImageList | ils | ilsAllIcons |
| TreeView | tre | treOrganization |
| Toolbar | tlb | tlbActions |
| TabStrip | tab | tabOptions |
| StatusBar | sta | staDateTime |
| ListView | lvw | lvwHeadings |
| ProgressBar | prg | prgLoadFile |
| RichTextBox | rtf | rtfReport |
| DTPicker | dtp | dtpDate |

* Choosing Prefixes for Other Controls
* For controls not listed above, you should try to standardize on a unique two or three character prefix for consistency. Use more than three characters only if needed for clarity
* For derived or modified controls, for example, extend the prefixes above so that there is no confusion over which control is really being used. For third-party controls, a lower-case abbreviation for the manufacturer could be added to the prefix. For example, a control instance created from the Visual Basic Professional 3D frame could use a prefix of fra3d to avoid confusion over which control is really being used
* Suggested Prefixes for Data Access Objects

|  |  |
| --- | --- |
| **DAO Type** | **Prefix** |
| Workspace | wrk |
| Database | dbs |
| container | cnt |
| Querydef | qdf |
| Recordset | rs |
| Tabledef | tbf |
| Field | fld |
| parameter | par |
| index | idx |
| user |  |
| group | grp |
| Environment | env |
| Connection | cn |
| Table | tbl |
| Resultset | rst |
| column | col |
| command | cmd |
| error | err |
| property | prp |

## Naming Variables and Functions/Routines

* Basic suffixes (%,&,#,etc.) should NOT be used as they can lead to non-intuitive code. Microsoft has also stated that the use of Basic suffixes should be discouraged
* The following base type prefixes should be used when naming variables and functions :-

| ***Data type*** | ***Prefix*** | ***Example*** |
| --- | --- | --- |
| Boolean | bln | blnFound |
| Byte | byt | bytRasterData |
| Collection object | col | colWidgets |
| Currency | cur | curRevenue |
| Date (Time) | dtm | dtmStart |
| Double | dbl | dblTolerance |
| Error | err | errOrderNum |
| Function | fun | funReverse |
| Integer | int | intQuantity |
| Long | lng | lngDistance |
| Object | obj | objCurrent |
| Procedure | pro | proclick |
| Single | sng | sngAverage |
| String | str | strFName |
| User-defined type | udt | udtEmployee |
| Variant | vnt | vntCheckSum |

## Scope Prefixes

* Scope prefixes are mandatory and should come before type prefixes
* The following table shows the naming of the same variable in different scopes :-

|  |  |  |
| --- | --- | --- |
| ***Scope*** | ***Prefix*** | ***Example*** |
| Global | g | gstrUserName |
| Module-level | m | mblnCalcInProgress |
| Local to procedure | None | dblVelocity |

## The Body of Variable and Routine Names

* Variable/routine names should be in mixed case, without underscores
* E.g. gnLastErrorStatus

## Naming User Defined Type

* User defined types should be declared with the first character in upper case, with subsequent words separated with underscores.
* E.g.

Type Cust\_details

szName As String

cSalary As Currency

End Type

* When declaring an instance variable of a user defined type, add a prefix to the variable name :-

Dim custNew as Cust\_details

## Understanding the Scope of Variables

* The scope of a variable defines which parts of your code are aware of its existence. When you declare a variable within a procedure, only code within that procedure can access or change the value of that variable; it has a scope that is local to that procedure. Sometimes, however, you need to use a variable with a broader scope, such as one whose value is available to all the procedures within the same module, or even to all the procedures in your entire application. Visual Basic allows you to specify the scope of a variable when you declare it
* You cannot declare public variables within a procedure
* Procedure-level variables are recognized only in the procedure in which they're declared. These are also known as local variables. You declare them with the Dim or Static keywords
* For example: Dim intTemp As Integer or Static intPermanent As Integer
* Values in local variables declared with Static exist the entire time your application is running while variables declared with Dim exist only as long as the procedure is executing
* Local variables are a good choice for any kind of temporary calculation. For example, you can create a dozen different procedures containing a variable called intTemp. As long as each intTemp is declared as a local variable, each procedure recognizes only its own version of intTemp. Any one procedure can alter the value in its local intTemp without affecting intTemp variables in other procedures
* By default, a module-level variable is available to all the procedures in that module, but not to code in other modules. You create module-level variables by declaring them with the Private keyword in the Declarations section at the top of the module
* For example: Private intTemp As Integer
* At the module level, there is no difference between Private and Dim, but Private is preferred because it readily contrasts with Public and makes your code easier to understand
* To make a module-level variable available to other modules, use the Public keyword to declare the variable. The values in public variables are available to all procedures in your application. Like all module-level variables, public variables are declared in the Declarations section at the top of the module
* For example: Public intTemp As Integer

1. **Comments and Headers**

## Comments

* Code and variables in the body of a procedure/function should be commented where their use is not obvious. If the comment extends over more than one line, a box comment should be used
* E.g.

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

' This is a large comment which

' extends over more than one line

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* REM statements should not be used

## Headers

* Visual Basic applications contain forms and modules that are held under version control. This requires the presence of a header in each form and module. Each header should contain the edit history of the form/module, including a reference to any procedures/functions/objects that have been changed. Each procedure/function in a module must have a header that gives a brief description of the procedure/function's action
* When PVCS is to be used for source management, the following should also be added to the header :-

‘Module Edit History:' $Log$

* Following constant should be declared following the header :-' Constant used for PVCS
* Const PVCS\_INFO = "$Revision$ READINI.BAS $Date$ $Author$"
* The following headers should be used in all VB applications :-
  1. ***Application Header***
  + The About form should contain the Application Header block

**'**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

' Application : RIGPA

'

' Application Description : LAN database and LAN manager

'

' Application Edit History :

' Version Date Changed By Forms/Modules Affected

'

' 1.0a 20Apr94 Farah Qureshi **All**

**'**

' Reason: First version

'

' 1.1a 25Jul94 Martin McDonnell frmMain

' MOVEPC.BAS

' Reason: SCR 001

'

' Application development environment : Microsoft Visual Basic Version x.x

'

' Application run-time environment :

' IBM PS/2 (or 100% compatible) Personal Computer

' Microsoft DOS Version 5.0 or higher

' Microsoft Windows Version 3.1 or higher

'

**'** Installation files : VBRUN300.DLL

' RIGPA.EXE

'

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + 1. ***Form Header***

The Form\_Load event should contain the Form Header block.

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

' Form : frmMain.frm

'

' Form Description : Main window for RIGPA

'

' Form Edit History :

' Version Date Changed By

'

' 1.0a 20Apr94 Farah Qureshi

'

' Change : First version

'

' 1.1a 25Jul94 Martin McDonnell

'

' Change : MouseMove event altered to include cmdStopPC

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + 1. ***Module Header***

The General Declarations section should contain the Module Header block.

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

' Module :' MOVEPC.BAS

'

' Module Description :' Procs/Functions needed to move a PC around quadrants

'

' Module Edit History :

'

' Version Date Changed By

'

' 1.0a 20Apr94 Farah Qureshi

'

' Changes in : All

'

' 1.1a 25Jul94 Martin McDonnell

'

' Changes in : RestrictMove

' nChangePCOwner

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + 1. ***Procedure Header***

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

' Procedure : RestrictMove

'

' Procedure Description : Stops a PC from moving to an occupied/reserved position

'

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + 1. ***Function Header***

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

'

' Function : nChangePCOwner

'

' Function Description : Changes a PC's owner number

'

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **Menus**

## Setting the Caption Property

* When assigning captions for menu items, you should try to follow these guidelines:
* Item names should be unique within a menu, but may be repeated in different menus to represent similar actions
* Item names may be single, compound, or multiple words
* Each item name should have a unique mnemonic access character for users who choose commands with keyboards. The access character should be the first letter of the menu title, unless another letter offers a stronger mnemonic link; no two menu titles should use the same access character. An ellipsis (…) should follow names of commands that require more information before they can be completed, such as commands that display a dialog (Save As…, Preferences…)
* Keep the item names short

## Menu Naming Conventions

* To make your code more readable and easier to maintain, it's a good idea to follow established naming conventions when setting the Name property in the Menu Editor. Most naming convention guidelines suggest a prefix to identify the object (that is, mnu for a menu control) followed by the name of the top-level menu (for example, File). For submenus, this would be followed by the caption of the submenu (for example, mnuFileOpen)

1. **Custom Controls**

## Custom Controls and Add-Ins

* Microsoft has licensed some custom controls for inclusion in VB Professional. These custom controls cover both specialized functionality that it would be unreasonable to expect VB to support, and also replacements for "weak" VB controls. The following issues should be considered when developing VB applications :-
* Vendor strength and support. Most custom control vendors do not have a UK presence, they are also often very small companies
* Quality/robustness of custom controls varies enormously
* Future compatibility. There is no guarantee that existing custom controls will be developed for use with future versions of VB
* Because of these issues, it is a good idea to consider most non-specialized custom controls as tactical, in that they compensate for functionality that is poor / missing in the current version of VB, but which we expect will improve in future versions

1. **Common dialogs**

* The CommonDialog control acts as an interface between your application and comdlg32.dll , the Dynamic Link Library that generates the dialogs for Windows 95.Using the CommonDialog control , you can display the following dialog boxes:
* Open
* Save As
* Print/Print Setup
* Color
* Font
* Help

1. **Validation and Testing**

## Validations

* Validations will be specific to a project. However this section gives guidelines for some common validations
* Length of the field : This can be done at design time by setting the MaxLength property of text boxes
* Checks for entry of mandatory fields should be done at Save
* Application specific validations, which are to be done on individual fields, should be done
* in Validate event of the control. However care should be taken that the program does not
* go in a loop of error messages
* If there is any processing going on use vbHourglass

## Testing

* An application should only be tested after it has been packaged as a VB independent application, i.e. as a .exe file Testing includes Forms, Active X DLLs, Active X EXEs,Active X controls and Active X documents. GUI of forms should be in compliance with the Windows Standards Active X controls and Active X DLLs should be tested by adding another Standard EXE Project to the Project Group. Testing of Active X EXEs should be done by creating another instance of Visual basic. Windows API calls should be made with the use of API Viewer

1. **Help**

* The Help for each screen should be displayed on click of the Help button. However, if the application so requires it, then context sensitive help can be provided. This should be done by setting the HelpContextID property of the individual controls to the respective topic's context id. Help, can then be invoked by pressing the F1 key

1. **Features of Visual Basic 5.0**

## MicroHelp

* Micro help should be displayed on GotFocus Events of the object
* All the micro helps should be written in one module only
* Micro help should be shown at the bottom of the screen on PANEL
* Font type of micro help should be BOLD and size should be 8.25 and font name should be MS-SansSerif
* Proper case structure should be used. (e.g. only 1st character should be in capital)

## Features of Visual Basic 6.0

* There are few major changes in Visual Basic 6.0 over Visual Basic 5.0, ranging from new language features to a re-vamping of the Setup Wizard (now called the Package and Deployment Wizard). ADO, or ActiveX Data Objects, has been implemented in VB6 as an ActiveX control (the ADO Data control). The third-party tool Crystal Report has been replaced by a more powerful tool Data Report
* The major changes are listed below:

## Objects

* Objects like File System Objects, or FSO, which include a set of methods that can make working with files and directories easier and faster
* The Enhanced version of CreateObject function can be used to create objects on remote systems by specifying an optional machine name according to DCOM specification

## Functions

* Functions like Round, CallByName, Filter, Format Currency, Format Date Time, Format Number Format Percent, InstrRev, Join, Month Name, Replace, Split, StrReverse, WeekDayName etc can be used to make the application development easier

## Properties and Methods

* Public properties and methods can use UDTs (user-defined types) as arguments or return value types. Under certain conditions, you can assign the contents of one array to another

## Database Access

* Database accessing method (ADO) can be used, which allows faster and flexible access to multiple data providers. ActiveX Data Objects (ADO) is the object-based interface to OLE DB

## Stored Procedures, Triggers

* Stored procedures, which are stored in the database, are written from the front-end instead of writing them in the back-end, which also applies for triggers. The On the Fly and Autolist features of Visual Basic also applies when writing stored procedures and triggers on the VB editor

## Making It Web Enabled

* Web enabled projects including smart Web pages are developed by taking the support of DHTML(Dynamic HTML) and IIS(Internet Information Server) server applications

## Data Reports

* Crystal reports might be avoided by introduction of Data Report, which makes it easier to produce reports. You can directly drop fields into the report from selecting it from the data environment
* The following issues should be considered when using Data Reports :-
* Before going for the Data Reports you have to set the Data Environment properties along with the Connection properties. More than one command should be added to the connection to take care if columns are selected from more than one table. If stored procedures are selected then special care should be taken when providing the in and out parameters
* You have to drag the columns from the Data Environment design window and drop it on the Data Report window Or Use ActiveX Data Objects (ADO) (extra code has to be written for the same action)