

---

# Infosphere Coding Standards

**Version 3.2**

**Prepared by Neville Silverman**

## Table of Contents

<b>Section</b>	<b>Page</b>
<b>1. Introduction</b> .....	<b>3</b>
Universal Standards .....	3
Why Have Coding Standards? .....	3
Naming Conventions .....	3
Enforcement .....	3
<b>2. Variable Scope Prefixes</b> .....	<b>4</b>
Scope Prefix Examples .....	4
Scope Example .....	4
<b>3. Constants</b> .....	<b>5</b>
Some examples .....	5
<b>4. Variable Naming Conventions</b> .....	<b>6</b>
Declaring Variables .....	6
Describing Variable Names .....	6
Naming Examples .....	6
Code Example .....	7
Naming Variables - Some DON'T's .....	7
Naming Variables - Some DO's .....	7
<b>5. Procedure Naming Conventions</b> .....	<b>8</b>
Naming Procedures - Some DO's .....	8
<b>6. User-Defined Types</b> .....	<b>9</b>
Example .....	9
<b>7. Object Naming Conventions</b> .....	<b>10</b>
Prefixes for Controls .....	10
Choosing Prefixes for Other Controls .....	11
Note .....	11
<b>8. Prefixes for Data Access Objects (DAO)</b> .....	<b>12</b>
Prefixes for DAO .....	12
An example: .....	12
<b>9. Prefixes for Menus</b> .....	<b>13</b>
Some examples .....	13
<b>10. Database Naming Conventions</b> .....	<b>14</b>
General Rules .....	14
Database Names .....	14
Database Names Examples .....	14
Table Names .....	14
Column Names .....	15
Foreign Key Example .....	15
Note .....	15
Access Object Name Prefixes .....	16
Other Database Object Prefixes .....	17
SQL Statements .....	17
<b>11. Comments</b> .....	<b>18</b>
Comment Examples .....	18
Some Guidelines .....	19
<b>12. Error Handling</b> .....	<b>20</b>
General Error Handling .....	20
Raising Errors .....	20
User Generated Errors .....	20
<b>13. Error Logging</b> .....	<b>21</b>
Properties .....	21
Procedures .....	21
Coding .....	21
Error Display .....	21
<b>14. Modular Programming</b> .....	<b>22</b>
Guidelines .....	22
Control Example .....	22

Function Example.....	23
<b>15. Good Programming Practices.....</b>	<b>24</b>
Some DOs.....	24
Some DONTs.....	24
Style.....	24
<b>16. Programming with Modules.....</b>	<b>25</b>
Creating Functional Modules.....	25
COM Objects.....	25
Calling Module Procedures.....	26
Using Local Variables.....	26
Using Enums.....	27
Using Properties.....	27

## 1. Introduction

---

These notes formalise the Infosphere coding standards when using VB (Visual Basic), or VBA (Visual Basic for Applications). They apply whether setting up a Microsoft Access Database, or when using VB to access any other Database project (i.e. Oracle, Sybase, SQL Server, etc).

### Universal Standards

There is an attempt to be universal, and the standard should allow, for example, the migration of a Microsoft Access Database to a back-end Server Database with minimal conversion problems.

### Why Have Coding Standards?

The main reason for using a consistent set of coding conventions is to standardise the structure and coding style of an application so that the programmer and others can easily read and understand the code. As maintenance costs are high, it is essential to write code that can be easily deciphered and enhanced by anyone in the Company.

Good coding standards result in precise, readable, and unambiguous source code that is consistent and as intuitive as possible. The object is to make the program easy to read and understand without cramping the programmer's natural creativity with excessive constraints and arbitrary restrictions.

A comprehensive coding standard encompasses all aspects of code construction and, while the programmer should exercise prudence in its implementation, it should be closely followed.

Using solid coding techniques and good programming practices plays an important role in software quality and performance. By consistently applying a well-defined coding standard and proper coding techniques, a team of programmers working on a software project is more likely to yield a software system that is easier to comprehend and maintain.

### Naming Conventions

Objects, Constants and Variables require well-formed naming conventions. This note lists recommended conventions for these, and discusses the issues of identifying data type and scope.

### Enforcement

These standards have been endorsed by Infosphere. They provide important benefits and must be implemented wherever practical. For implementations which build on pre-existing software, it may be necessary to adopt practises already employed in that software or restrict these standards to new modules within the project.

## 2. Variable Scope Prefixes

---

As the project size grows, so does the value of recognizing variable scope quickly. A one-letter scope prefix preceding the type prefix provides this, without greatly increasing the size of variable names.

### Scope Prefix Examples

Scope	Prefix	Example
Global	g	gUserName
Module-level	m	mFirstTime
Local to procedure	None	ClientName

Consistency is crucial to productive use of this technique.

### Scope Example

Here is an example of how to code variables with Module scope:

Option Explicit

```

Dim mInsertRecord As Boolean      '-- Set when a Row is added
Dim mGridArray() As Variant      '-- Holds the Data
Dim mClassGrid1 As clsTrueGrid   '-- Reference to Grid Class
Dim mMaxRows As Long             '-- Number of Rows to use
Dim mMaxColumns As Integer       '-- Number of Columns setup
    
```

### 3. Constants

---

There is a generally accepted VB standard that constants be declared in upper case. They should be prefixed with the Scope identifier. As with all programmer defined names, no underscore (\_) should be used. This is reserved for system Constants.

#### Some examples

```
'-- Constants are Private by default.  
Const ROWMAX = 100  
  
'-- Declare Public constant in the modGlobal Module.  
Public Const gHEADING = "Client Name"  
  
'-- Declare Private Integer constant.  
Private Const CLIENTCOLUMN As Integer = 5
```

## 4. Variable Naming Conventions

---

For the computer, a unique name serves only to differentiate one item from another. However, expressive names function as an aid to the human reader; therefore, it makes sense to provide a name that the human reader can fully comprehend.

### Declaring Variables

Declaring all variables saves programming time by reducing the number of bugs caused by typing errors. On the Editor tab of the Options dialog, make sure that the Require Variable Declaration option is checked. The Option Explicit statement requires that you declare all the variables in your VB program. Unfortunately this option is by default disabled.

This is also applicable to Access programming (i.e. using VBA). Thus for an Access Module, you **MUST** see

```
Option Compare Database
Option Explicit
```

### Describing Variable Names

The variable name should have mixed case and should be as long as necessary to describe **completely** its purpose. It should be obvious from the variable name, as to what data type (i.e. String, Boolean or Numeric) is being used. Make the variable name long enough to be meaningful but short enough to avoid being wordy.

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.

Minimize the use of abbreviations. Make sure that the abbreviations are consistent throughout the entire application. Randomly switching between Cnt and Count within a project can lead to unnecessary confusion. However, rather than abbreviating the word Count, be more descriptive, for example use PageCount or RecordCount.

### Naming Examples

- ❑ Examples of String variables are SQL, ClientName, ClientInitials, ClientAddress.
- ❑ Numeric variables should be equally clear, i.e. BytePosition, FileLength, RowTotal.
- ❑ Boolean variables should obviously be either True or False, for example FirstTime or AllOK or IsValid. If there is any doubt, add the prefix "Is".
- ❑ Date/Time variables should indicate what their contents, i.e. EntryDate (d/m/y), EntryTime (h:m:s), EntryDateTime (d/m/y h:m:s).

## Code Example

Here is an example of how to code variables:

Option Explicit

```
Dim mInsertRecord As Boolean '-- Set when a Row is added
Dim mGridArray() As Variant '-- Holds the Data
Dim mClassGrid As clsTrueGrid '-- Reference to Grid Class
Dim mMaxRows As Long '-- Number of Rows to use
Dim mMaxColumns As Integer '-- Number of Columns setup
```

## Naming Variables - Some DON'T's

- ❑ Don't use a prefix to indicate the variable's data type – this is an old and dreadful standard. Instead, make sure that the variable name is completely meaningful, and makes the data type obvious. This is much more important than using a prefix for the data type.
- ❑ Don't use a "-", this should be reserved for Events of controls, and System variables.
- ❑ Don't declare variables on the same line. This does not allow descriptive comments on each variable.
- ❑ Don't use cryptic names for a variable. The names X1 and X2 are not acceptable, except to a mathematician doing complex matrix manipulation.
- ❑ Don't use "i"s and "j" for subscripts. Use meaningful names like BytePosition and RowIndex.
- ❑ Don't use abbreviations, where for a few extra characters, you can be more precise. Abbreviations are to be used only for long words.
- ❑ Don't use typographical marks to identify data types, such as \$ for strings or % for integers.

## Naming Variables - Some DO's

- ❑ Do indicate the scope of a variable with Global and Module prefixes.
- ❑ Do rename all variables, after having completed coding, until they are completely meaningful.
- ❑ Even for a short-lived variable that may appear in only a few lines of code, still use a meaningful name.

## 5. Procedure Naming Conventions

---

The name of the Function or Sub routine should contain mixed case characters, and should be as long as necessary to describe **completely** the purpose of the procedure. In other words, make the name long enough to be totally meaningful.

### Naming Procedures - Some DO's

- ❑ Do use the verb-noun method for naming Functions and Sub routines, such as CalculateInvoiceTotal() or InitializeNameArray().
- ❑ Do make the purpose of Procedure name clear, i.e. CalculateTotalSalaries, OpenCustomerDatabase, ReadCustomerRecordset.
- ❑ Do include a description of the value being returned for string Functions, such as GetCurrentClientName().

## 6. User-Defined Types

---

In a large project with many user-defined types, it is often useful to give each such type a three-character prefix of its own. In addition, a suffix of "Type" must be used. For example, "cli" could be used as the prefix for variables of a user-defined Client type.

### Example

```
Type ClientType
  cliName      As String
  cliAddress1  As String
  cliAddress2  As String
  cliAddress3  As String
  cliCity      As String
  cliPostCode  As String
  cliState     As String
End Type

Dim Client as ClientType

Client.cliName = "Tom Jones"
```

## 7. Object Naming Conventions

Objects should be named with a consistent prefix that makes it easy to identify the type of object. Recommended conventions for some of the objects supported by VB are listed below.

### Prefixes for Controls

Control type	prefix	Example
3D Panel	pnl	pnlGroup
ADO Data	ado	adoBiblio
Animated button	ani	aniMailBox
Check box	chk	chkReadOnly
Combo box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Communications	com	comFax
Control	ctr	ctrTextBox
Data	dat	datBiblio
Data grid	dgd	dgdTitles
Directory list box	dir	dirSource
Drive list box	drv	drvTarget
File list box	fil	filSource
Flat scroll bar	fsb	fsbMove
Form	frm	frmEntry
Frame	fra	fraLanguage
Gauge	gau	gauStatus
Graph	gra	graRevenue
Grid	grd	grdPrices
Horizontal scroll bar	hsb	hsbVolume
Image	img	imgIcon
ImageList	ils	ilsAllIcons
Label	lbl	lblHelpMessage
List box	lst	lstPolicyCodes
ListView	lvw	lvwHeadings
MAPI message	mpm	mpmSentMessage
MAPI session	mps	mpsSession
MCI	mci	mciVideo
Menu	mnu	mnuFileOpen
Month view	mvw	mvwPeriod
MS Chart	msc	mscSalesbyRegion
MS Flex grid	msg	msgClients
MS Tab	mst	mstFirst
Object	obj	objMyClass
OLE container	ole	oleWorksheet
Option button	opt	optGender
Picture box	pic	picVGA
Picture clip	clp	clpToolbar
ProgressBar	prg	prgLoadFile
RichTextBox	rtf	rtfReport
Shape	shp	shpCircle
Slider	sld	sldScale

<b>Control type</b>	<b>prefix</b>	<b>Example</b>
StatusBar	sta	staDateTime
SysInfo	sys	sysMonitor
TabStrip	tab	tabOptions
Text box	txt	txtLastName
Timer	tmr	tmrAlarm
Toolbar	tlb	tlbActions
TreeView	tre	treOrganization
UpDown	upd	updDirection
Vertical scroll bar	vsb	vsbRate

### Choosing Prefixes for Other Controls

For controls not listed above, you should try to standardize on a unique three character prefix for consistency. Use more than three characters only if needed for clarity.

### Note

It is completely acceptable to use “obj” as a prefix for a generic Object, i.e. objTextBox or objControl or objDatabase.

## 8. Prefixes for Data Access Objects (DAO)

As with Control Object, a standard prefix is also required for DAO (Data Access Objects). Use the following prefixes to indicate DAO.

### Prefixes for DAO

Object	Prefix	Example
Container	con	conReports
Database	DB	DBAccounts or objDatabase
DBEngine	dbe	dbeJet or objJetEngine
Document	doc	docSalesReport
Field	fld	fldAddress
Group	grp	grpFinance
Index	idx	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	RS	RSForecast or ForecastRS
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
User	usr	usrNew
Workspace	wsp	wspMine

### An example:

```
Dim DBBiblio As Database
Dim RSPubsInNSW As Recordset
Dim SQL As String
Const DBREADONLY = 4 '-- Set constant.

'-- Open the database.
Set DBBiblio = OpenDatabase("BIBLIO.MDB")

'-- Select the Recordset
SQL = "SELECT * FROM Publishers WHERE State = 'NSW'"

'-- Create the new Recordset object.
Set RSPubsInNSW = DBBiblio.OpenRecordset(SQL, DBREADONLY)
```

## 9. Prefixes for Menus

---

Applications frequently use many menu controls, making it useful to have a unique set of naming conventions for these controls. Menu control prefixes should be extended beyond the initial "mnu" label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string.

### Some examples

<b>Menu caption sequence</b>	<b>Menu handler name</b>
File Open	mnuFileOpen
File Send Email	mnuFileSendEmail
File Send Fax	mnuFileSendFax
Format Character	mnuFormatCharacter
Help Contents	mnuHelpContents

When this naming convention is used, all members of a particular menu group are listed next to each other in VB's Properties window. In addition, the menu control names clearly document the menu items to which they are attached.

## 10. Database Naming Conventions

---

It is common practice in Access to have a prefix of tbl. This is not adhered to in any Microsoft Access Database example (i.e. Northwind), or SQL Server examples or any other third party Server Database system. If there is no possibility of the Database being converted, the “tbl” prefix standard may continue to be used.

### General Rules

- ❑ There should not be spaces in a Column (Field), Control, or Object name. It is possible that spaces in names can produce naming conflicts in VBA. Also, spaces in field names don't always work in non Access Database platforms.
- ❑ The underscore (\_) should not be used as word separator. It can cause confusion with VBA routines.
- ❑ Use upper and lowercase designations in names, for example a Table called AccountsPayable.

### Database Names

The Database name should describe the functionality of the Database. It should not be the name of the Company or the Project.

### Database Names Examples

Accounts, Inventory, GeneralLedger

### Table Names

Table names, for Access Databases which may later be changed to say SQL Server, or VB+DB, should not contain a prefix of “tbl”. The name should always be in the singular, because a table usually holds more than one record, so it is plural by implication. For example, use Customer, not Customers.

Choose the Table name carefully - since changes to the names of Access objects do not propagate through the Database. For example, changing the name of a Table late in the development cycle requires changing all the Queries, Forms, Reports, Macros, and Modules that refer to that table.

Examples of Table Names: Customer, OrderDetail, Supplier

## Column Names

Column Names (i.e. Fields of Tables) must be descriptive. Although the name can be up to 64 characters long, generally 15 characters should be sufficient. Use a mixture of upper and lower case.

- ❑ All Primary Indexed Fields must have a suffix of ID. This includes Primary Keys and Foreign Keys (including Drop Downs). And of course, all Tables must have a Primary Key.
- ❑ All Field names should be prefixed with a three-character, lower case abbreviation of the parent Table. For example for a Customer Table, it may have fields called `cusCustomerID`, `cusAddress`, `cusStateID`.
- ❑ Where a Master/Detail relationship exists, a four-character prefix should be used. This will comprise of the Master (or Header) Table prefix, and the character "d". For example for a Bank Header Table and Bank Detail Table, the prefixes used will be "bnk" and "bnkd" respectively.
- ❑ Do not repeat the table name; for example, avoid having a field called `empEmployeeLastName` in a table called `Employee`.
- ❑ For Fields that have a common meaning throughout the Database, no prefix should be used. This allows for easy recognition of the fields, and allows handling by standard sub routines. Examples are `ModifiedBy`, `ModifiedDateTime`, `Status`.
- ❑ Field names should indicate the form of Date/Time used. Examples are `StartDate` (d/m/y), `StartTime` (h:m:s), `StartDateTime` (d/m/y h:m:s).
- ❑ Use the Description property when adding each Field.
- ❑ Set the Caption field before building any Forms.

## Foreign Key Example

Table:	Customer	State
	<code>cusCustomerID</code>	<code>staStateID</code>
	<code>cusStateID</code>	<code>staDescription</code>
	<code>cusAddress</code>	<code>staCode</code>
	<code>cusPostCode</code>	
	<code>ModifiedByUser</code>	
	<code>ModifiedDateTime</code>	
	<code>Status</code>	

## Note

There are two prevalent conventions for Field names, with and without a prefix. In VB+DB, where the Database access routines are completely under the control of the programmer (i.e. using DAO or ADO), the prefix is not needed – but is still commonly used.

For Microsoft Access, the prefix eliminates possible confusion between Tables with common field names, and is highly desirable.

## Access Object Name Prefixes

Other than for Tables, all other Access Database Container Objects must have a name prefix.

A suffix qualifier is used at the end of a Form or Report name for a SubForm or SubReport. If the Form frmProductSupplier has a related SubForm, it should be called frmProductSupplierSub. This allows the object and its SubForm or SubReport to sort next to each other in the Database container.

<b>Object</b>	<b>Prefix</b>	<b>Example</b>
Query	qry	qryOverAchiever
Form	frm	frmCustomer
Sub Form	frm	frmCustomerSub
Report	rpt	rptInsuranceValue
Sub Report	rpt	rptInsuranceValueSub
Macro	mac	macUpdateInventory
Module	mod	modBilling
Class	cls	clsADO

The programmer should name each Database object that refers to a table with the same base name as the table, using the appropriate prefix to differentiate them. For example, if the table is Customer, its primary Form would be frmCustomer, and its primary Report would be rptCustomer.

## Other Database Object Prefixes

There are three additional Database object prefixes:

- ❑ "zz" is used for objects that are temporary or not in use, but that the programmer may want to keep in the Database for future reference or use (for example zzfrmPhoneList).
- ❑ "zs" denotes system objects (for example the zsControl Table). System objects are items that are part of the development and maintenance of an application not used by end users, such as error logs, development notes, documentation routines, relationship information, and so on.
- ❑ "\_" denotes objects under development (for example, \_macNewEmployee). Remove the underscore when the object is ready to use and it will sort normally.

These prefixes cause the system objects to be sorted either toward the bottom or the top of the Database container. For example, a prefix of "zz" causes the object name to sort to the bottom of the Database container, where it's available but out of the way. An underscore before an object name sorts to the top of the Database container, to visually remind the programmer that it needs attention.

## SQL Statements

- ❑ When writing SQL statements, use all uppercase for keywords and mixed case for database elements, such as Tables, Columns, and Views.
- ❑ Put each major SQL clause on a separate line so statements are easier to read and edit, for example:

```
Dim SQL As String

SQL= SQL & "SELECT FirstName, LastName "
SQL= SQL & "FROM Customers "
SQL= SQL & "WHERE State = 'WA' "
```

- ❑ Use SELECT \*. Network traffic is minimally increased, but Database processing is more efficient (only exception is with large Images). The programmer will be less likely to make errors, and any additional Columns added will always be available.

## 11. Comments

Comments are invaluable to the programmer who must maintain any intricate software. Comments in a source code listing should allow the code to stand on its own, without external documentation. The Comments must be maintained and updated in parallel with source code development.

If you write more than 10 lines without a Comment, your code is badly written!

### Comment Examples

- At the beginning of every Form or Module, it is helpful to provide boilerplate Comments, showing the routine's author, purpose, and modifications.

```
'--
'-- *      Routine      : Error Handling Module
'-- *      Written by:  N. Silverman
'-- *      Date         : 21/3/2001
'--
'--
'--                               Modifications
'-- Date           Programmer      Summary of Changes
'-- -----
'-- 25/5/2001     Nev              Mods per User request 20/5/2001
```

- At the beginning of the start-up Form, show the References and Components selected.

```
'-- Reference: Microsoft ActiveX Data Objects 2.5 Library
'-- Component: Sheridan DataGrid/Combo/DropDown 3.1 (OleDb)
```

- Any changes to code must have a Comment on the line changed, the date, the reason and by whom.

```
'-- 02/06/02 Nev - Corrected divide by zero
```

- At the beginning of every Procedure, it is helpful to provide Comments indicating the routine's purpose.

```
'-- This routine will append all errors to Error.log
'-- in the chosen directory.
```

Where the Procedure is important and critical, use the following format:

```
*****
' Purpose : Opens the Connection to SQL Server
' Inputs  : The Database Name
' Defaults: DBLock = adLockOptimistic
'          DBCursor = adOpenDynamic
'          DBOpenMode = adModeReadWrite
'          SQLServerProvider = "SQLOLEDB"
'          ServerName = "Server"
'          DBUser = "sa"
'          DBPassword = ""
' Returns : Boolean depending on success of the Connection
*****
```

- ❑ Add end-line Comments to explain Variable declarations. Align all end-line Comments at a common tab stop.

```
Dim mInsertRecord    As Boolean      '-- Set when a Row is added
Dim mGridArray()    As Variant      '-- Holds the Data
Dim mClassGrid1     As clsTrueGrid  '-- Reference to Grid Class
Dim mMaxRows        As Long         '-- Number of Rows to use
Dim mMaxColumns     As Integer      '-- Number of Columns setup
```

- ❑ Use Comments to explain the intent of the code. Comment anything that is not readily obvious in the code.

```
With Grid
    .DataMode = ssDataModeAddItem  '-- Add Item mode
    .RemoveAll                      '-- Removes all Rows
    .Columns.RemoveAll              '-- Removes all Columns
    .FieldDelimiter = vbTab         '-- Must not be in Text
    .FieldSeparator = ", "
    .BackColorOdd = &HFFFFFF        '-- Remove the Blue Rows
    .ForeColorEven = &H80000012     '-- Make the row black
End With
```

### Some Guidelines

- ❑ Separate Comments from Comment delimiters, i.e. use something like "-- ". Doing so will make the Comments stand out and easier to locate.
- ❑ Prior to deployment, remove all temporary or extraneous Comments to avoid confusion during future maintenance work.
- ❑ Use complete sentences when writing Comments. Comments should clarify the code, not add ambiguity.
- ❑ Comment as you code, because most likely there won't be time to do it later. Also, should you get a chance to revisit code you've written, that which is obvious today probably won't be obvious six weeks from now.
- ❑ When modifying code, always keep the Comments around it up to date.

## 12. Error Handling

---

The programmer needs to add error handlers to every procedure where there is the slightest possibility of an error. Where the display of a MsgBox is not desirable, the Err.Raise method should be used.

### General Error Handling

This should be the standard for all Error handling:

```
Private Sub MyDescriptiveProcedureName ()
    On Error GoTo ErrHandler

    DoSomething

ErrExit:
    '-- Put in any cleanup logic
    Exit Sub
Errhandler:
    MsgBox "MyDescriptiveProcedureName: " & Err.Description
    Resume ErrExit
End Sub
```

### Raising Errors

There are some occasions when it is not desirable to show an Error Message. In these cases, the Error may be returned to a higher level calling routine.

```
Private Sub ErrorRoutine(ProcedureName As String)
    '-- This will immediately returns an Event to the caller program

    Err.Raise Err.Number, , "nsAdo: " & ProcedureName & ": " _
        & Err.Source & ": " & Err.Description
End Sub
```

Note that the Error Message returned provides a full description of the origin of the error.

### User Generated Errors

If the programmer needs to exit a routine with a generated error condition, special care should be taken not to use an existing VB error number. This can complicate error debugging. To use the number 1052, add the VbObjectError constant - this will ensure that the Error Number used is unique.

```
Err.Number = vbObjectError + 1052
```

## 13. Error Logging

The module modErrorLog provides a consistent standard for error logging and display. It should be imported into each system that is developed.

### Properties

- BatchMode            Stops the Error Message display
- ErrorFileName        Sets the log file Name and path. Default is AppPath\Error.log
- UserContact           Sets the user contact name and particulars in the Error message display.

### Procedures

- MsgLog                Displays the error message. This would normally only be called from the User Interface.

### Coding

By adding the following line to the Error Handling routine, a MsgBox can be generated and any errors Logged.

```
modErrorLog.MsgLog "cmdError_Click", Me.Name, Err.Description, _
Err.Number
```

### Error Display

This is an example of the standardised Error Message.



This is what appears in the Error Log

```
2001:09:16 10:33:14 Release: 001:000:000 frmTestErrors
cmdError_Click 6 Overflow
```

## 14. Modular Programming

---

**M**odular programming means the breaking down of the code into “Black Boxes” using meaningful, easily comprehensible and easily debugged procedures. This helps to keep maintenance times down to minimum.

### Guidelines

- ❑ Ideally, each procedure should be no more than 1 screen in length.
- ❑ Any code greater in length than 1 screen, should be broken into a control routine with multiple called Functions.
- ❑ Where a Method is important and critical, it should be placed in a Function, by itself, with precise handling of all error conditions.

### Control Example

This is the “Mainline” controlling the logic flow. It is equivalent to pseudo code, each step being obvious from the name of the called Function.

```

Private Function ControlRoutine() As Boolean
    On Error GoTo ErrHandler
    If Not OpenDataBase Then
        GoTo ErrExit
    End If
    If Not ReadRecordset Then
        GoTo ErrExit
    End If
    If Not ValidateData Then
        GoTo ErrExit
    End If
    CleanUp
    ControlRoutine = True
ErrExit:
    Exit Function
ErrHandler:
    ErrorRoutine "ControlRoutine: "
    Resume ErrExit
End Function
    
```

## Function Example

This is an example of a called Function. It is a critical Function, and would be used multiply – hence the need to ensure adequate error handling and Function identification.

```
Public Function DBExec(SQL As String, Count as Integer) As Boolean
'*****
' Purpose : Runs and Executes an SQL statement
' Inputs  : String containing the SQL statement
' Returns : The number of Record affected
'*****
    On Error GoTo ErrHandler
    '-- Execute the SQL command
    adoConnection.Execute SQL, Count, adCmdUnknown
    DBExec = True
ErrExit:
    Exit Function
ErrHandler:
    ErrorRoutine "DBExec: " & SQL
    Resume ErrExit
End Function
```

## 15. Good Programming Practices

---

**W**ithout duplicating all VB coding manuals, here are some guidelines on programming essentials for trouble free coding, and minimising maintenance efforts.

### Some DOs

- ❑ Keep the lifetime of variables as short as possible when the variables represent a finite resource for which there may be contention, such as a database connection.
- ❑ Keep the scope of variables as small as possible to avoid confusion and to ensure maintainability. The potential for inadvertently breaking other parts of the code can be minimized if variable scope is limited.
- ❑ Be specific when declaring objects, and use the full Library Name (i.e. ADODB), instead of just the object (i.e. Recordset), to avoid the risk of name collisions. You will also have the advantage of using the Auto List Members facility.

```
Dim mEmployeesRS      As ADODB.Recordset
Dim mobjDocument      As Word.Document
```

- ❑ Use early binding techniques whenever possible. Again you will also have the advantage of using the Auto List Members facility.
- ❑ Use Select Case statements in lieu of repetitive checking of a common variable using If...Then statements.
- ❑ Explicitly release object references at the first opportunity. For example, close ADO Recordset and Connection objects to insure that connections are promptly returned to the connection pool for use by other processes.

```
Set mEmployeesRS = Nothing
```

### Some DONTs

- ❑ Don't use the GoSub...Return Statement. Creating separate procedures that you can call provides a more structured alternative to using GoSub...Return.

### Style

- ❑ Take the time to ensure that all If/Else/End If statements are correctly aligned
- ❑ Use a standard size of four spaces for the indent.

## 16. Programming with Modules

It is common practice to lump all odd procedures into Modules, usually called Module1.bas, Module2.bas, etc. This adds nothing to the simplification of logic, nor does it aid in providing the code with self-documentation.

### Creating Functional Modules

Each Module should have only procedures relating to the functionality of the Module, and all related logic should be in the same place – i.e. the Module should “encapsulate” all the logic. Use only Boolean Functions. Here is a typical structure for a large project:

Module Name	Description
modGlobal	Contains <b>all</b> Global Variables and Global constants. These would have a prefix of “g”. All other Modules should only have Private (or Local) variables.
modADO	Contains all logic relating to ADO handling.
modDAO	Contains all logic relating to DAO handling.
modErrorLog	Error handling routines
modEdit	Contains all string handling functions.
modAPI	Contains all Windows API functions
modRules	Contains all Business Rules

You can't have enough Modules!

### COM Objects

With the Modules structured into discrete entities, it is relatively easy to create COM Objects (i.e. a DLL) from them.

Module Name	Description
modGlobal	Contains all Global Variables and Global constants.
modAPI	Contains all Windows API functions

  

DLL Classes	Description
clsADO	Contains all logic relating to ADO handling.
clsDAO	Contains all logic relating to DAO handling.
clsErrorLog	Error handling routines
clsEdit	Contains all string handling functions.
clsRules	Contains all Business Rules

#### Notes

- ❑ The Classes must be Instantiated in order to be used.
- ❑ The Classes have Initiation and Termination Events. This allows more control than for a Module.
- ❑ The Classes can have collections.

### Calling Module Procedures

When calling a procedure in a Module, the Module name should be used as a prefix, i.e. `modErrLog.MsgLog`. The advantages of doing this are:

- ❑ It provides better documentation
- ❑ "Intelligence" is available to display all the Modules procedures
- ❑ Private procedures are "hidden"
- ❑ It avoids conflicts between similarly named procedures
- ❑ Functions are better defined, i.e `modDAO.GetRecordset` and `modADO.GetRecordset` provide precise descriptions of the procedures being used.

Of course all Public procedures in each Module are available to the project as a whole without a prefix. This may save a few seconds of coding time, but it is bad practice.

### Using Local Variables

As all logic is encapsulated in a Module, the use of Local (i.e. Private) Variables becomes a particularly powerful tool.

For example:

```
Dim mIsDatabaseOpen as Boolean

Public Function GetRecordset(RS as ADO.Recordset) as Boolean
    If not mIsDatabaseOpen then
        MsgBox "The Database has not been opened"
        Exit Function
    End If

End Function
```

## Using Enums

Where a variable has different numeric values, representing different conditions, it helps with the documentation to declare the different values as a global Enum. This is a better practice than using Constants.

As an example, if the State variable can be coded with 1 for NSW, 2 for VIC, etc – this is how it should be handled:

```

modGlobal
Public Enum StateEnum
    NSW = 1
    VIC = 2
    NT = 3
    TAS = 4
    SA = 5
    WA = 6
End Enum

Form
Private Sub MyProcedure()
    Dim State As StateEnum

    State = NSW
    MsgBox State    '-- The MsgBox will return the value 1
End Sub
    
```

When “State =” is entered, all values of the Enum (NSW, VIC, etc) are displayed for selection. This is exactly the method used in VB or VBA for any Global constant.

Note that this feature is only available for variables with numeric constants.

## Using Properties

The Property feature allows complete control of the setting and retrieval of Variables in a Module. In this example, the Database name can be set and retrieved.

```

Option Explicit

Private mvarDatabaseName As String    '-- Holds the property value

Public Property Let DatabaseName(ByVal vData As String)
    if not CheckPath(vData) then
        MsgBox "The Database path is invalid"
        Exit Property
    End If
    mvarDatabaseName = vData
End Property

Public Property Get DatabaseName() As String
    DatabaseName = mvarDatabaseName
End Property
    
```

By removing either the “Get” or “Let” procedure, the programmer can control access. There is complete control of what values are set, i.e. the check to see if a valid file path has been provided. The values set can be modified in the Property procedure (say providing a default path for the database) and errors returned, if necessary.

Note that all code logic would only use the mvarDatabaseName variable.