E contents of VB

A Definition

Visual Basic is a much-enhanced version of the BASIC programming language and the BASIC Integrated Development Environment (IDE). The bottom line of the enhancement is the VB can create Windows programs whereas BASIC could only create DOS programs. Ok, so the modifications are **very** major, but the idea holds true that Visual Basic is BASIC for Windows.
One of the many significant improvements is that VB provides massive support for easily creating the user interface to your applications. This is accomplished within the VB Integrated Development Environment (IDE), in which you use a mouse to "draw" your application and use the keyboard to type in the code that is to be executed. When I write a VB program, I almost always create the user interface "shell" before I write any code at all. This approach, made so simple by the VB IDE, allows me to evaluate how the user will interact with the program. I can generally create the initial version of the shell in just hours. It's a much better way to program than to create pieces of the interface (and the corresponding code) as you go. It especially is beneficial in that you can demonstrate to the users just exactly what they will see, and you can do it early enough in the development cycle to prevent costly reiterations later in the cycle.
The single largest effect on coding that VB introduced was the concept of an event-driven programming model. In the old BASIC you had to write code to watch for the occurrence of user events (pressing a key, using the mouse, ...). VB performs that function for you, and in fact, the **only** time code will execute in VB is in response to such an event!
And finally, the other major concept that VB has incorporated is the concept of **objects**. Objects provides a way to link together both code and data into a "package" in such a way as to make handling and saving the code/data more intuitively. VB forms are objects, menus are objects, and the so are the intrinsic VB controls. A lot more on this later!
VB has also provided a wide variety of built-in code that programmers once had to handle themselves. Of most significance is the built-in database handling features of VB. It is generally accepted that over half of all VB applications are written to handle databases! You'll find the built-in database features of VB to be very powerful, and that you can tap into them at whatever level of programming skill you possess.
A second area in which VB has begun to provide built-in support is that of Internet access. The VB features are still maturing, but with the tools available you can create very useful applications. I've included Internet topics in the Advanced section of the tutorial because it is such a specialty application, not so much because its concepts cannot be handled by less skilled programmers. Feel free to skip around in the tutorial if you find a section that interests you.

Critical Visual Basic Elements

Although Visual Basic has grown into a fairly complex programmming tool, it is still the case that a programmer can pretty much ignore all of the capabilities he doesn't need (or understand) and still create very useful applications.
But, no matter how much a beginner, or how advanced you are, there are still some fundamental areas in which you must be proficient to become a VB programmer.
When you start VB, you will see a group of windows that are know as the VB IDE (integrated development environment). As a programmer you will spend the majority of you time here, so you might as well get used to the IDE and spend a fair amount of time exploring the menu options that the IDE provides. Pay closed attention to the keyboard shortcuts that are available. If you've read my other Beginner sections you'll know that I am a **big** fan of the keyboard and have argued that the #1 productivity tool you have is good typing skills. While in the IDE, you'll find the keyboard shortcuts invaluable in writing your program quickly.
The second aspect of VB which programmers of all skill levels will have in common is the VB language itself. Most of the questions I get through email are focussed on "how do I ..." and the

answer is almost always couched in terms of the code that it takes to perform the task. If ever there was less boring reading than the VB language reference manual, I don't know what it i

but it also provides one of the biggest payoffs of studying that I can recommend. At least 90% of every question I answer for visitors to my site **is in the manual!**

Every Visual Basic application will consist of controls, usually a lot of them! In my opinion, the availability of controls (built-in, or controls you can purchase) is the single biggest reason why VB has reached the level of popularity that it currently enjoys. Because controls represent hundreds (if not thousands) of hours of manpower to come up with full-featured, debugged code which you can resuse in your program, controls are easily the most cost-effective, and the most time-effective way that a VB programmer has to add features to his program. Bottom line is that any good programmer must be an expert at handling controls. In my experience, once you've mastered the intrinsic controls, learning new controls becomes very straightforward.

Because VB has moved to the event-driven model of programming, the last critical VB topic I will mention is that of events. Events are not very complicated but the concept is significantly different than the old-style linear programming of the original BASIC. Simply put, when a VB program is started it sits and waits for an event to occur. The event can be a keypress by the user or the movement of a mouse. Either way, the VB programming model is that your program will only react to events. When an events occurs, VB will execute the code associated with that event. So your job as a programmer is to basically create the code which your program executes in response to those events.

Programming Basics

To toss in a bit of philosophy, I view a programmer's job as getting to the end product with a minimum of effort. When you're paid by the hour you have an obligation to your employer (even if it's yourself) to minimize the cost/schedule of completing the assignment. To that end, I will emphasize over and over again the importance of using existing capabilities (such as your own reusable code, VB-provided controls, or even controls that you purchase).

In light of this optimize-your-time philosophy, I also believe that every VB programmer should strive to become an expert in the following two areas: Databases and Reporting. It's very common for applications to store data and to provide that data in a printed format (by some estimates, 80% of all VB applications use databases). And while this tutorial will discuss various ways to perform these tasks, most programmers will find that the capabilities of VB for creating/editing Access (a Microsoft product) databases and VB's capabilities for reporting that data (as exemplified by the Crystal Reports control or the newer built-in VB reporting features) are the most effective tools you can use for increasing your effectiveness in creating VB applications.

If you want to become a serious VB programmer, you **must** become an expert in these areas. I'm not saying that Microsoft's built-in tools are the only, or best, tools available for creating/editing databases and reporting on them. What I am saying is that tools which provide similar functions will be one of the most commonly used tools of those available to a programmer and that any serious programmer must take the time to develop strong skills with these tools.

One by-product programming strategy of my philosophy is that newbies should not get too anxious to jump into the more advanced features of VB. Take the time to learn the basics, and especially how to apply them with ingenuity. Once you've exhausted the potential of the fundamentals is the time to consider more advanced techniques. In my experience, over 90% of my programs consists of fundamental programming techniques, while only rarely am I compelled to dip into the more complex features that VB has to offer.

VB History (VB3, VB4, VB5 and VB6)

Starting with VB5, Visual Basic became an exclusively 32-bit programming language, suitable for programming only Win9X or NT systems. If you must program for Win 3.x, then you'll have to drop back to either VB3 or VB4, both of which are in pretty short supply. VB4 had the dual ability to support Win3.x as well as Win9X/NT systems but my personal

recommendation is that if you need 32-bit system support, go straight to VB6 and if 16-bit is your need then stick with VB3.

The VB Learning Edition is the most affordable, and truth is that you can do a lot with it, particularly if you use the Windows API to augment its capabilities. However, in light of its better database features and its greater variety of controls, I suggest you go straight to the Professional Edition if at all possible. The price is steep, but it really does pay itself back in terms of time savings. If you need the VB Enterprise edition then you should have it paid for by the "Enterprise" which requires it. Individual programmers generally do not need the Enterprise edition.

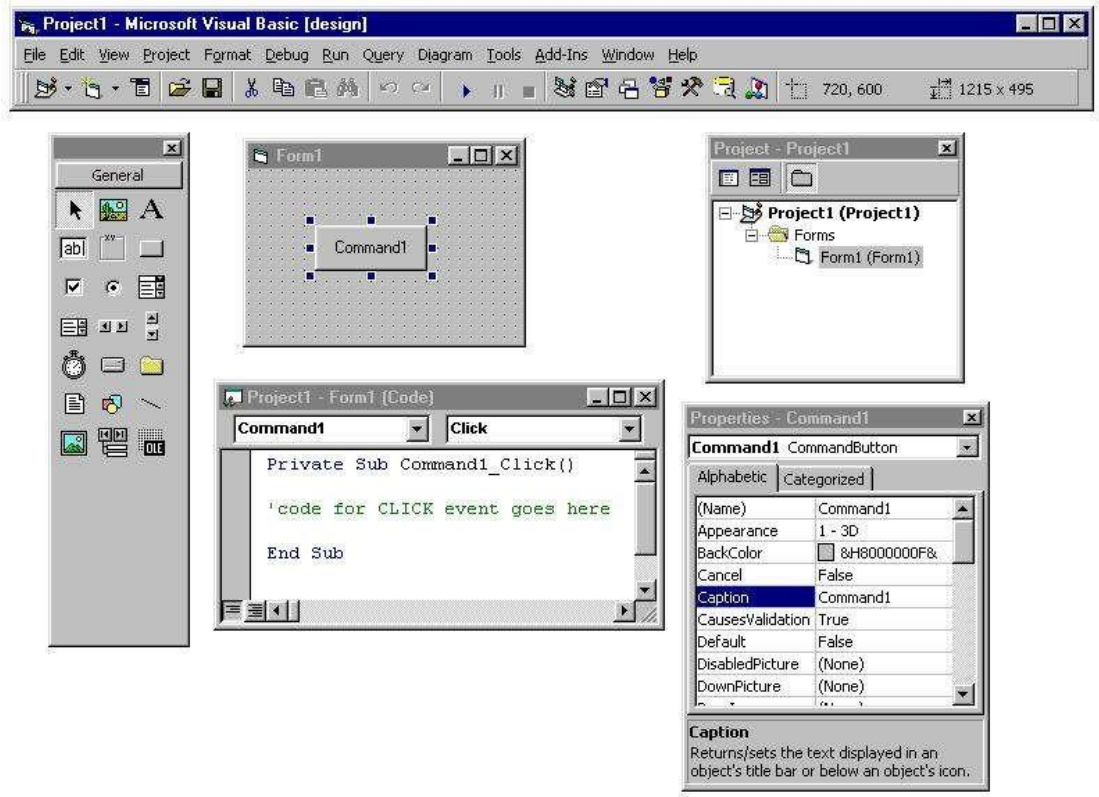
IDE

It's a little known fact, but you can write a VB program from scratch using nothing more than a simple text editor, such as the Edit or Notepad applications which come with Windows. In fact, the Visual Basic project files are exactly that - text files. However, writing a project from scratch would involve a **lot of tedious, detailed** manual entries. To simplify the task, Microsoft has built in to VB a software program to help you write your VB projects. That software, known as the Integrated Development Environment (IDE for short) is what comes to the screen when you start VB and is the topic of this section.

Overview

Like any other Windows application, VB consists of multiple windows which appear at startup. The windows that are displayed when you start VB are collectively known as the Visual Basic Integrated Development Environment (IDE).

When you first start VB all of the windows are locked together in what is called the MDI format. I prefer the SDI format (which you can set in the options menu) which allows each of the windows to be positioned independently on your screen. Here's a sample IDE screen which shows a VB project with one form on which is a single command button.



In particular, VB has the following windows:

Menu / Toolbar

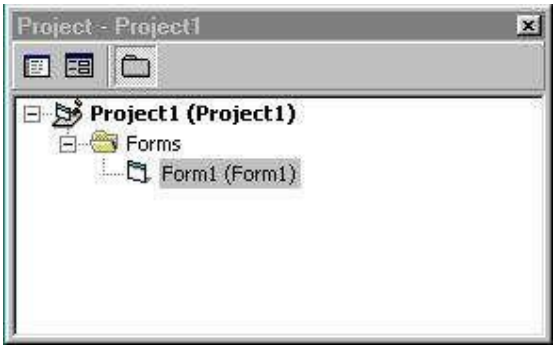
This is the only element of the IDE which is always visible. You use it to select which other IDE elements to view and to add forms or controls to your project. There are many other features which we will discuss later.

Toolbox

The toolbox is simply a library of controls which you can place on your application. Once you've placed all the controls you need onto your applications forms, you can hide the toolbox to make room for working in the other elements of the IDE.

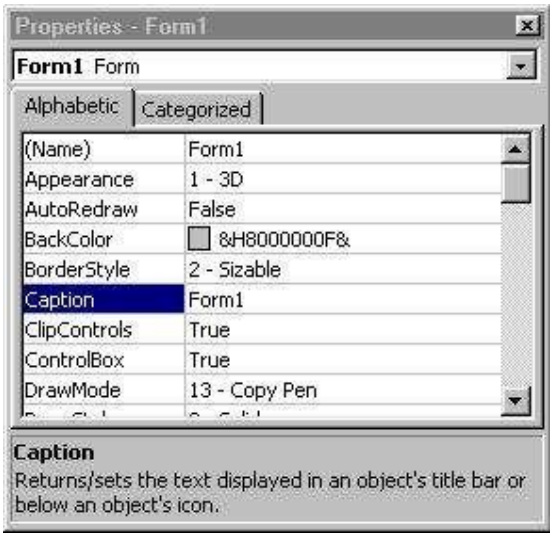
Project Window

This is simply a list of all the forms which make up your VB project. There are several kinds of forms which we'll talk about later.



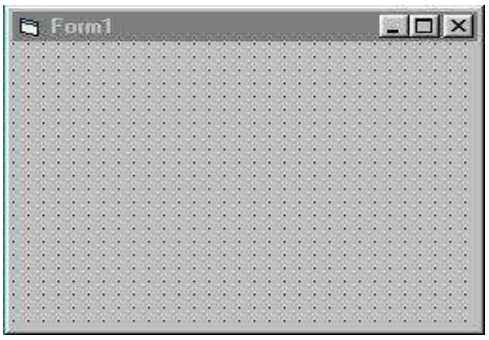
Property Window

We'll talk about controls later, but such things as push-buttons, scrolling text boxes, pictures boxes and other features of most VB applications allow you to enter parameters which define how these controls work. In VB, these parameters are called properties. Some properties can be entered at design time within the IDE, while others must be entered with code while the program is running.



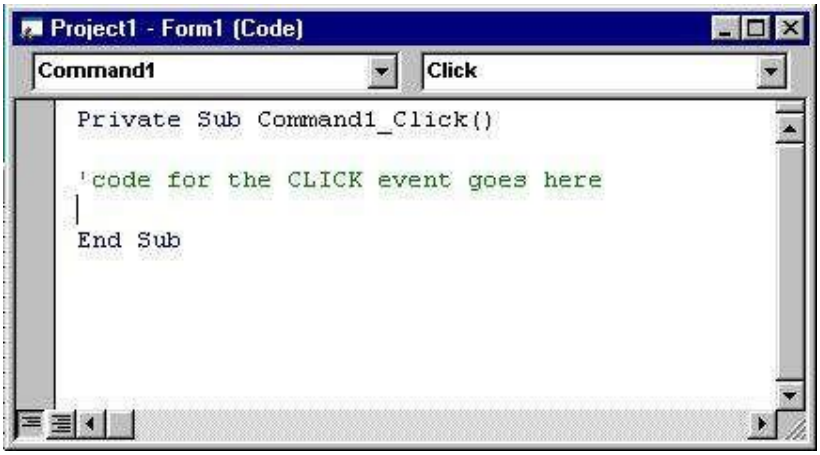
Forms

You add these to your VB application as they are needed. They are the windows which hold the various controls (buttons, text boxes, etc.) which make up your application.



Code Window

Like it's name implies, this is where you type in the code that VB executes. Notice that the heading of the window indicates with which event the code is associated.



A VB Session

When VB begins a new project, it starts with a single form which has no controls placed on it. A
session in VB would go something like this:

1. Add additional forms (if needed)
2. Set form properties

3. Place controls on forms

4. Set control properties
5. Write code for form/control event procedures
6. Create executable (.EXE) or simply RUN program within the IDE
7. Save project

It's important to understand that each form in VB is saved as a standalone file on your computer. The listing of all forms, and their location on your hard disk, are kept in another file which is called the project file. In VB3 the project file extension was .MAK but in VB6 it is .VBP. This file is just an ASCII text file which provides information about the VB project. There is a variety of other information which is held within the file, but all of it is directed towards describing those files (and controls) which make up the VB project. We'll talk more about the content of the file in other sections of this tutorial. As your skills improve, you'll even feel comforable about editing the project directly, using a simple text editor such as notepad. I do it occasionally when I want to make a simple change to a program without going through the VB IDE. Even more useful is that if I want to capture a routine that I know is in a VB project, I can call up the file that has the code and copy it to the clipboard for insertion into my new project. This has the benefit of being very quick and it lets me use search tools that are not a part of the VB IDE.

Your First Application

Ok, it's already time to create you first application. Beginner's are usually shown what's called a "Hello World" application. It's called this because all the application does is display the words "Hello World" when the user presses a button. As useful applications go, it's not much, but it does show what is involved in creating an application.

Step 1.
Start VB. As it VB comes up, it automatically creates a NEW application consisting of only 1 form which has no controls.

Step 2.
Using the mouse, move the cursor over the toolbox and click with the left mouse on a command button control. This selects the button. Now move the cursor to the form and while pressing the left mouse button draw out a rectangular shape with the mouse. Once the shape is drawn and the left mouse is released, a command button appears on the form.

Step 3.
Change the CAPTION property of the command button. Do this by clicking once on the button to select it, then pressing F4 to bring up the Property window. Scroll in the window until you find the CAPTION property. Highlight the CAPTION property and type in "PRINT". The text on the command button will be replaced by what you type.

Step 4.
Remember earlier in the tutorial that we talked about writing code for events? Well, each of VB controls recognize certain events. In this example, the CLICK (left mouse press and release) is the event which corresponds to pressing the button.

VB uses a nomenclature such as command1_click() for event procedures. Since we've not yet discussed procedures this information is a bit premature. For now, just take on

faith that VB provide a place to write the code which will be executed when the button is clicked.

To get to the location where the code will be placed, DOUBLE-CLICK on the button. A new windows, the CODE WINDOW, will appear with the cursor in place, ready for you to type the code. Type the following line of code:

```
PRINT "Hello World"
```

Step 5.

Your program is now complete and ready to run. Within the IDE you simply press F5 to
run the program. Do so now.

Your application will appear as a single window in which there is a single button labeled "Print". Press the button with your mouse and you will see the words "Hello World" appear in the upper left corner of the window.
That's it! You've just written your first VB application.
To return to the IDE, click on the "X" button at the top right of the form. Later we'll discuss better ways to exit from a program.

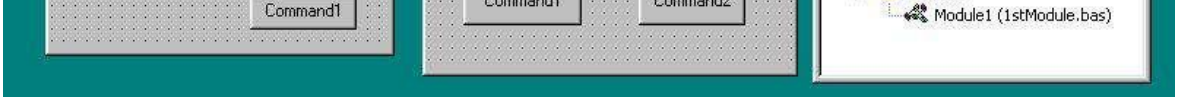
Projects

The end result of most VB programming efforts is an executable program (one that has a .EXE extension). This file is compiled from the actual text files which make up a VB project. VB doesn't put the entire project into a single file. Instead, it allows the programmer to break up a project into several smaller files. Each of these files can be used in more than one VB project. The group of files used to compile the application is called a VB project and is the topic for discussion in this section of the tutorial.

Project Definition

A typical VB application might consist of more than one forms, each of which may have multiple controls. In VB, the information about each form (it's own properties as well as those of the controls that are on the form) is saved into it's own file. In fact, there are several types of files which VB creates (we'll cover them all later in the tutorial).
VB also saves a "project" file which contains the list of files which VB loads when the "project" is loaded. The project file includes other information, such as the filenames of controls which are not intrinsic to VB (they exist as separate files with .OCX extensions).
The project file is a simple ASCII text file and may be edited with any text editor, including NOTEPAD (which comes with Windows). Once you learn the format of the project file you may have reason to edit it directly but in general, most programmers do their editing within the VB IDE.
The project file is saved with an extension of .VBP (Visual Basic Project). Other files saved by VB include the following extensions:
- .FRX : graphics associated with the form
- .BAS : code not associated with control events

The good thing about splitting a project into many files is that you can use a file (form or otherwise) in one or more projects. If you make a change to the one file, all using projects see the change.
In the following VB IDE example, you see a project with two forms, each with multiple controls. The filename of each form is displayed in the project window, as is the name of a single .BAS file. Together, all three files make up the VB project.



Here is the content for each of the files. You will note that the files are in ASCII text. By inspection, you can guess what a lot of the line items are for.

Project File

```
Type=Exe
Form=2ndForm.fr
m
Reference=*\G{00020430-0000-0000-C000-
000000000046}#2.0#0#..\WINDOWS\SYSTEM\STDOLE2.TLB#OLE Automation
Form=testform.frm
Startup="Form1"
Command32=""
Name="Project1"
HelpContextID="0"
CompatibleMode="0
" MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="Personal
" CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
Retained=0
ThreadPerObject=0
MaxNumberOfThreads=1
```

Form 1

```
VERSION 5.00
Begin VB.Form Form1
Caption =
"2ndForm"
ClientHeight = 2280
ClientLeft = 5805
ClientTop = 2055
ClientWidth = 3735
LinkTopic =
"Form1"
ScaleHeight = 2280
ScaleWidth = 3735
Begin VB.CheckBox Check2
Caption =
"Check2" Height =
495
Left = 2160
TabIndex = 3
Top = 360
```

Width = 1215
End
Begin VB.CheckBox Check1
Caption =
"Check1" Height =
495

Left = 240
TabIndex = 2
Top = 360
Width = 1215
End
Begin VB.CommandButton Command2
Caption =
"Command2" Height =
495
Left = 2160
TabIndex = 1
Top = 1320
Width = 1095
End
Begin VB.CommandButton Command1
Caption =
"Command1" Height =
495
Left = 360
TabIndex = 0
Top = 1320
Width = 1215
End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId =
True Attribute VB_Exposed =
False Option Explicit

Form 2

VERSION 5.00
Begin VB.Form Form2
Caption = "Form2"
ClientHeight =
1905
ClientLeft = 6600
ClientTop = 2025
ClientWidth = 3465
LinkTopic =
"Form2"
ScaleHeight = 1905
ScaleWidth = 3465
Begin VB.CommandButton Command1
Caption =
"Command1" Height =
375
Left = 2040
TabIndex = 2
Top = 1320
Width = 1095
End
Begin VB.TextBox Text2

Height = 285
Left = 1080
TabIndex = 1
Text =
"Text2" Top =
720
Width = 1695

```
End
Begin VB.TextBox Text1
Height = 285
Left = 1080
TabIndex = 0
Text =
"Text1" Top =
240
Width = 1695
End
End
Attribute VB_Name = "Form2"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId =
True Attribute VB_Exposed =
False Option Explicit
```

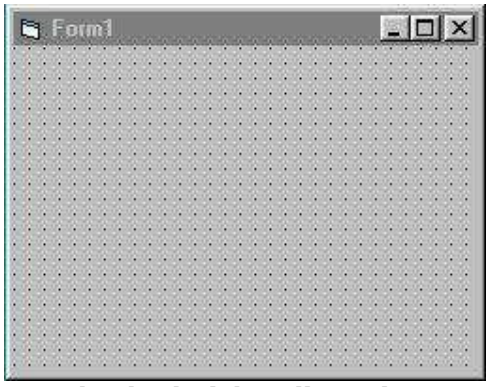
As you go through more of the tutorial, the content of the files will make more sense, but even now you can see that the information is not that difficult to decipher.

Forms

Visual Basic forms are windows. It's an important piece of data because it ties the concept of a form in with everything you already know about Windows applications. These rectangular shaped areas of the computer screen are called windows and the whole strategy of the Windows Operating System is to manage the display of those windows (while running the code that generates them or which performs calculations in the background). Since you've already been exposed to other Windows programs, then you already intuitively understand the concept of a form (window)! This section provides additional details about how VB handles forms.

What is a Form?

Here's a simple Visual Basic form. It looks just like any other form that you use in Windows applications. The header area has a caption, the control menu, and the minimize/maximize/close buttons. The the large area of the form is called the client area.



Don't be shocked, but all Windows/NT programs consist of one or more windows. In its simplest form, a window simply consists of a rectangular area of the screen. Anything that appears inside that area is considered to be part of the window. However, you can have one window contained inside another. Control objects, which are also implemented as windows, will be framed by the form window to which it belongs. As an operating system, Windows 9.X/NT controls the display of the various, possibly overlapping, windows on the screen. In Visual Basic, the basic building block of an application is a form, which is simply a window. The VB IDE can insert forms into your project, and then you can resize the forms as well as change other properties of the form.

However, controls (checkboxes, textboxes, ...) are also windows. A form is distinguished from a control in that only forms can exist as standalone objects. When controls are used, they must be placed in a form. Ok, there are a few exceptions such as the printer object or the screen object

which are not considered part of any form, but are part of a VB program. I'll talk to these special
"system" objects later in the tutorial.
Not to confuse the issue, but controls can also be placed inside of other controls. When this happens the parent control is known as a container. Likewise, forms are containers but are the highest level of container there is in a windows application. Forms are always parents of controls, never the other way around.
There is one exception which I will not cover in these tutorials, and that is a special form called an MDI form. In this special case, an MDI form is always contained within a parent form. This is exactly the same type of parent/child relationship which you see in Word. Each new Word document is contained in its own window, but is always framed within the larger window that is the Word application.
The MDI (multiple document interface) forms can be very useful in applications where multiple files/images/documents need to be open at the same time. Other than this brief mention, I will not cover MDI forms in these tutorials.

Properties / Events / Methods

Now is a good time to bring up the 3 categories of information which may be used to describe any object, including forms. Forms, like any object, have properties which you may set. The properties range from the caption that the form displays to the physical size of the form. Later on this page I list all of the properties/events/methods that a form recognizes.
Likewise, a form may recognize certain events. All forms recognize the same events, but there are controls which recognize a broader range of events than forms. Events range from a simple keypress by the user to the click of a mouse button.
Then, finally, forms and controls also support various actions that may be taken. The actions are
known as methods, and may include such tasks as moving the form, loading it into memory, or refreshing the form to redraw graphics which may have been overshadowed when one form was placed on top of another.
Remember that even though this part of the tutorial is focussing on forms, that Properties/Events/Methods apply to **all** objects in Visual Basic.
Given that there are over 20 controls available to you in the VB Pro edition, you might be concerned that learning all of the possible properties, events, and methods could be an overwhelming task. However, it's not at all that bad. Here's a very helpful piece of information that makes your task easier: **all forms, controls, or objects share many of their properties, events, and methods!**. Re-read what I just wrote! It's a very important piece of information and it means you can reuse what you learn about one control to help you learn about other controls. I've created a **control summary chart** (available in **Excel97** and **Excel5.0** formats. The chart
gives the complete list of VB controls (provided in VB Pro) and lists their properties, events, and methods. The chart is listed in such a way that you can see the common items, as well as those which are unique to that control. You'll see that many controls have no unique items at all! In my chart I show 41 common properties, 20 common events, and 7 common methods. Please note
that not every control uses all the common items! Some common items may be shared by only 2 or 3 controls.
I highly recommend that you look over the chart and become familiar with **all** of the items on it. I regularly get questions at my site where the programmer could have performed a desired task by simply setting a property of the control, **if he had just known about it!** Just having the chart gives you the ability to look and see if a control supports a feature that you need.
Detailed descriptions and sample code for using the items can be found in the VB HELP file. There is also a Microsoft book call "The VB6 Language Reference" which gives additional detail for each of the items.

As a prelude to the larger chart, here's a simple listing of the entire set of properties, events, and methods that are recognized by a form. Don't shy away from looking at this list in detail because you **will** use every one of these over and over again! Forms are particularly critical to the VB programmer because they are the fundamental building block for **every application**.

Properties	**Events**	**Methods**
Name	Click	Refresh

Appearance	DragDrop	Drag
BackColor	DragOver	Move
BackStyle	GotFocus	SetFocus
BorderStyle	KeyDown	ZOrder
Caption	KeyPress	OLEDrag
CausesValidatio	KeyUp	ShowWhatsThi
n Container	LostFocus	s
Enabled	MouseDow	
Font	n	
ForeColor	MouseMove	
Height	MouseUp	
HelpContextID	OLEDragDrop	
hWnd Left	OLEDragOver	
MaskColor	OLEGiveFeedBac	
MouseIcon	k OLESetData	
MousePointe	OLEStartDrag	
r	Validate	
OLEDropMod		
e Parent		
RightToLeft		
Style		
Tag		
Text		
Top		
Visible		
WhatsThisHelpI		
D Width		

You'll note that forms, like most objects tend to have many properties, and that the number of events is much larger than the number of methods. A control can have well over 100 properties but normal count is usually around 50-70. You'll find that the name of most of the properties to be very self-explanatory (i.e., caption, name, fontsize, enabled, ...).
On the other hand, controls are not likely to have more than 25 or so events, and rarely has more than 10 methods. There are exceptions, but the generalization gives you a feel for what is involved with most controls.
The **controlsummarychart** lets you quickly get to details of which controls supports which item.

Controls Overview

Quick! How many controls come with VB? Can you list them? Well, it's not as easy as you might think. The documentation that comes with VB is a bit vague on exactly what is available to you. Even worse, I've found multiple places in the documentation which don't even say the same thing! Finally, not all that's on the CD-ROM gets installed on your PC, so you can't simply use the Components dialog box to determine every OCX control that VB has to offer! Because
of that, I've created this part of the tutorial. I still have a separate section for Intrinsic (built-in) and ActiveX (separate OCX files) controls.

Introduction

When I was writing the sections on the Intrinsic and ActiveX controls, I realized that the documentation for the controls was somewhat confusing. The MS Programmers Guide, the MS Component Tools Guide, and the VB HELP files weren't consistent in their description of what controls were available, nor were they clear as to which controls come with the Learning and Professional editions of VB.

Intrinsic Controls

In the next section of the tutorial, I cover the 20 controls which are built in to VB. These controls

are called intrinsic controls. Every one of the intrinsic controls are available to every VB program you write. When you create a "New" VB project, all 20 of the intrinsic controls will show up in the Toolbox. This is true for all versions of VB.

ActiveX Controls

Microsoft uses this terminology to refer to any control which exists as a separate file whose extension is OCX. To use an ActiveX control in your VB program, the control must be registered in the Windows Registry. Usually, the OCX control installation software handles the registration for you (such as VB does to register the controls it provides).

If you have a control which has not been registered by other software, you can register it yourself using the free program provided by Microsoft. The program, REGSVR32, comes with all

version of Windows and is usually found in the Windows folder. To use it, simply type in:

```
REGSVR32 control.ocx
```

where the "control.ocx" is the filename of the control that you want to register.

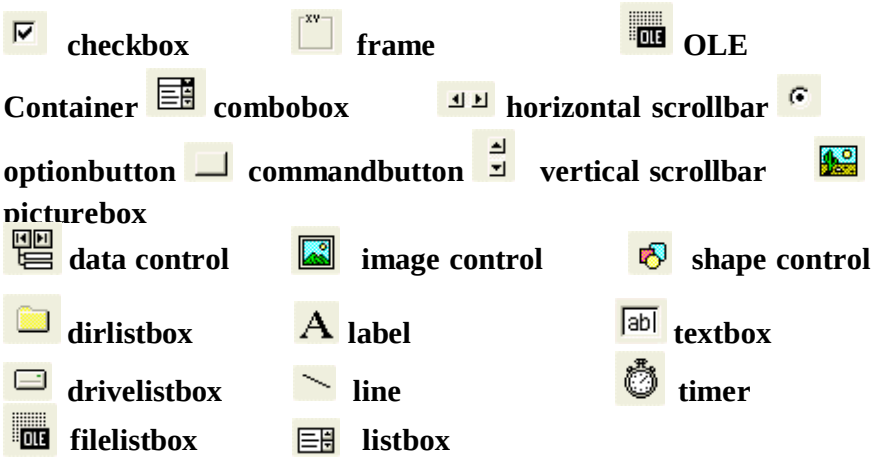
IDE and Controls

Here's a key point to remember. Just because you register a control **does not** mean that you can use the control in one of your projects. Registration only assures you that the control can by used by an application at run-time. Whether or not a control can be used at design-time (within the IDE) depends on the control.

There are many free controls which can be used freely at design-time. However, commercial controls require that you install them using a password before you can use them within your project at design time. If you simply copy an OCX to your system and register it with REGSVR32, don't be surprised if you get an error message when you try to put the OCX on a form!

Intrinsic Control List

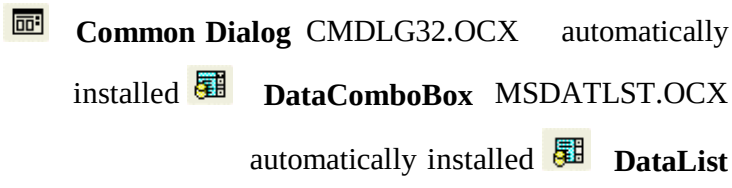
The following list shows the 20 intrinsic controls that come with all version of VB. I have a separate tutorial section to discuss them further:



Beyond the intrinsic controls, what you see and what you can install from the CDROM depend on which version of VB you've purchased. Not only that, but VB doesn't automatically install every possible OCX onto your system. Some OCXs (which can be used in the IDE) can be found on the VB CDROM. I'll show you where in just a minute!

Learning Edition ActiveX Control List

The next list shows the OCX controls which come with the VB Learning Edition. I show you which ones get automatically installed, and which ones you must manually install.



MSDATLST.OCX automatically installed **MSFlexGrid**

MSFLXGRD.OCX automatically installed

Professional Edition ActiveX Control List

These are the OCX controls which come with the VB Professional Edition. I show you which ones get automatically installed, and which ones you must manually install. In a later tutorial section I also discuss which controls are the most useful to programmers.

ADO Data Control MSADODC.OCX automatically installed

Animation Control MSCOMCT2.OCX automatically installed

Communications Control MSCOMM32.OCX automatically installed

CoolBar Control COMCT332.OCX automatically installed

DataGrid Control MSDATGRD.OCX automatically installed

DataRepeater Control MSDATREP.OCX automatically installed

DateTimePicker Control MSCOMCT2.OCX automatically installed

DBGrid Control DBGRID32.OCX automatically installed

DBCombo Control DBLIST32.OCX automatically installed

DBList Control DBLIST32.OCX automatically installed

FlatScrollBar MSCOMCT2.OCX automatically installed

Grid Control GRID32.OCX automatically installed

ImageCombo Control MSCOMCTL.OCX automatically installed

ImageList Control MSCOMCTL.OCX automatically installed

Internet Transfer Control MSINET.OCX automatically installed

ListView Control MSCOMCTL.OCX automatically installed

MAPI Controls MSMAPI32.OCX automatically installed

Masked Edit Control MSMASK32.OCX automatically installed

MonthView Control MSCOMCT2.OCX automatically installed

MSChart Control MSCHART.OCX automatically installed

Multimedia Control MCI32.OCX automatically installed

MSHFlexGrid Control MSHFLXGD.OCX automatically installed

	PictureClip Control	PICCLP32.OCX	automatically installed
	ProgressBar Control	MSCOMCTL.OCX	automatically installed
	RemoteData Control	MSRDC20.OCX	automatically installed
	RichTextBox Control	RICHTX32.OCX	automatically installed

Slider Control MSCOMCTL.OCX automatically installed **StatusBar Control** MSCOMCTL.OCX automatically installed **SysInfo Control** SYSINFO.OCX automatically installed **SSTab Control** TABCTL32.OCX automatically installed **TabStrip Control** MCSOMCTL.OCX automatically installed **Toolbar Control** MSCOMCTL.OCX automatically installed **TreeView Control** MSCOMCTL.OCX automatically installed

UpDown Control MSCOMCT2.OCX automatically installed

WinSock Control MSWINSCK.OCX automatically installed

Making Your Own Choices

I've shown you which controls are available but I've yet to tell you which controls are of any real use to you. You'll find that in applications you write, you will use a good sprinkling of the intrinsic controls. In my experience, about 90% of the controls on my forms have come from VB's intrinsic control list.

Once you select the intrinsic controls you need, you'll add to your project a few of the ActiveX controls which provide a specific feature you want in your application. You are very unlikely to have an application which uses all of the available ActiveX controls. You'll often find that adding 6-10 ActiveX controls will be the most that any application requires. If you're like me, you will have 3-4 very versatile controls which you use a lot, and a very few others which you use on a regular basis.

From the available ActiveX controls, I use the Toolbar, ImageList, and Common Dialog controls extensively. The Treeview, and Tab Control round off my list of personal favorites from the ActiveX control list. I probably should make more use of the Progress bar but most of my applications don't include tasks which make the user wait long enough to warrant the insertion of a Progress Bar. Likewise, using the StatusBar would certainly help my programs be more user- friendly, but I just don't seem to generate the enthusiasm to include it on my applications.

I also depend on third party OCXs. I find that I use the Formula One spreadsheet and the Crystal Reports reporting controls very often. Other than that I have a few specialty OCXs which I use, particularly in the graphics areas.

Instrinsic Controls

VB comes with 20 built-in controls. In this section of the tutorial I provide a few comments about each one, trying to give some useful pointers on the use of each control. I strongly suggest that you review my **control summary chart** - available in **Excel97** and **Excel5.0** formats. The chart gives the complete list of VB controls - along with their properties, methods, and events. The spreadsheet will help you get a "big-picture" overview of the VB controls. I also strongly suggest you read the HELP file content for each of the properties/events/methods for these controls. If you don't know the item exists, then you won't know when to apply it in your applications!

All controls are not equally useful. Some you will use on every application you write. Others you will use only when you have a special need for the features the controls offer.
When you start VB, you'll always find the intrinsic controls displayed in the toolbox. The controls are built-in to the VB files and do not exist in an external file (with a .OCX extension) the way the ActiveX controls do. In the toolbox, each of the controls has its own distinctive icon.

The Most Useful Intrinsic Controls

These nine intrinsic controls are pretty much used on every VB application I've written. Start your learning with these and then branch out. Further down on this page I have a brief comment on each of the controls.

Command Button Image

CheckBox PictureBox

TextBox ListBox

Label ComboBox

Option Button

The Rest of the Intrinsic Controls

The other eleven intrinsic controls are also valuable but I find myself using these less often than
the others. Also, you'll find that you use fewer of these within an application than you do of the nine that I listed as the most useful controls.

ADO Data Contro Vertical Scroll Bar

DirListBox Line

DriveListBox Shape

FileListBox OLE

Frame Timer

Horizontal Scroll Bar

Database Features

I've put the discussion of databases elsewhere in the tutorial, but you should know right now that
several of the intrinsic controls can display or edit data directly out of a database. With VB, the ADO Data Control is used to access the database information and to distribute it on to

the other intrinsic controls which can handle database information. VB uses the terminology "databound" to describe controls which have built-in features for handling database access.

Comments on Each Control

Command Button
This one works just like you expect. Press the button and it executes a block of code.

CheckBox
Typically this is used for turning on/off some particular feature of your program.

TextBox
This is the standard way of letting a user edit information. To make it convenient for the
users, learn about the .SELLENGTH and .SELSTART properties which highlight text in the the textbox.

Image
Use this to display a picture. Use it over the PictureBox because it takes less operating system resources.

PictureBox
While it can display pictures, it also acts as an area on which you can print text and graphics. Use it for home-grown graphics or print previews.

Label
As the name suggests, this is used to label other controls. It's pretty passive and you'll seldom use its items other than the .CAPTION property.

Option Button
If you use it, you'll use it in groups. VB handles the feature that only 1 option button can be selected at a time.

ListBox
This is the first of the intrinsic controls to introduce methods common to some of the more complex controls. The use of a ListIndex which starts at 0 (not 1) is a confusing factor that you must watch in your code.

ComboBox
Whereas a listbox takes up space on the form, the combobox control minimizes the use of
valuable form real estate. It has 3 modes of operation some of which allow you to keep your users from entering bad data.

ADO Data Control
If you're not accessing a database, then you don't need this one. If you are accessing a database you have to have this control to act as the interface to any other databound control. The exception is that VB offers ways to access databases directly from code, but for modest display/edit applications the ADO Data Control is very effective.

DirListBox / DriveListBox / FileListBox
You'll almost always use these in combination with each other. Read the HELP file for how to synchronize them to work together. Often, however, you will use the CommonDialog Control instead of these.

Line

It can be used as a static display, or you can animate it with the .VISIBLE property and the .MOVE method.

 **Shape**
When the line is not enough, this one supports rectangles and ellipses/circles. As with the
line control, use its items to create animation.

 **OLEContainer**
If you want to put objects on your VB application which come from other applications already on your machine (such as Word, Excel, ...) then this control is very useful. For my needs, I don't like making the assumption that my users have the application (in a specific version) on their machine to make distribution of my application go smoothly. I avoid this one whenever possible!

 **Frame**
It's just a container - it can hold other controls. There are two very good reasons to use it.
If you want multiple groups of option buttons then place each group in a frame and each group will operate independently. If you want to manipulate controls as a group (i.e., positioning) then put them in a frame and you can handle them all at one time.

 **Horizontal/Vertical ScrollBar Controls**
Basically you let use the slider value of a scroll bar as the input for other code that you write. These are normally used in conjunction with other controls.

 **Timer**
This is the most unusual of the intrinsic controls. By setting the .INTERVAL property this control will automatically create an event on a regular basis. No other control does this! You can use it to create an action at a certain time and then turn the control off to prevent repeats.

ActiveX Controls

In the Learning Edition of VB there were only 4 ActiveX controls, but in the Professional Edition of VB, Microsoft has provided 20 additional controls. Some are very excellent and some you may never use. In this section of the tutorial I provide a few comments about each one, trying to give some useful pointers on the use of each control. I strongly suggest that you review my **control summary chart** - available in **Excel97** and **Excel5.0** formats. The chart gives the complete list of VB controls - along with their properties, methods, and events. The spreadsheet will help you get a "big-picture" overview of the VB controls (both Intrinsic and Pro). Equally strongly, I suggest you read the HELP file content for each of the properties/events/methods for these controls. If you don't know that the property or method of an object exists, then you won't know when to apply it in your applications!

Introduction

As with the Intrinsic controls, not all of the ActiveX controls are equally useful. Some you will use on many applications but you will use others only when you have a special need for the features the controls offer.
When you start VB, **none** of the ActiveX controls are displayed in the Toolbox. Only the intrinsic controls are displayed, so you must manually insert the ActiveX controls into the Toolbox as you need them. To do so, right-mouse click on the toolbox and go to "Components", select the controls to put on the toolbox and press "OK".
A little tip - in the startup box for VB you can select to open a project which contains every ActiveX control that VB installed. You might have missed it because that option is at the

end of the list of project types that VB can open for you when you use the "File/New" menu. Here's a picture of the Toolbox with all of the Pro controls loaded.

All of the ActiveX controls are contained within OCX files. None are built-in to VB. In some cases, Microsoft put more than one control into an OCX file. We'll give the file names later.

The Most Useful ActiveX Controls

Of course, everyone will have their opinion but since it's my tutorial I get to give my own opinion for the most popular Pro controls. These nine intrinsic controls are pretty much used on every VB application I've written. Start your learning with these and then branch out. Further down on this page I have a brief comment on each of the controls.

Coolbar
Progress Bar
ImageList
SSTab
Internet Transfer Control
Tab Strip
ListView control
ToolBar
Multimedia control

The Specialty ActiveX Controls

Some of the VB controls are very excellent tools, but they simply aren't needed in many applications. This list of controls are ones that would be on my list of the best controls except that they are just not needed that often:

Communication control
RichTextBox
Data Repeater
SysInfo
MAPI Session
MAPI Messages
Winsock
MSChart

The Rest of the ActiveX Controls

Of the remaining ActiveX controls, I find some useful but mostly they sit un-used on my PC. It doesn't mean they don't perform their features well, it just means that my own applications don't find the need for the features very often.

Animation
MonthView
DateTimePicker
PictureClip
FlatScrollBar
Slider
ImageCombo
UpDown
MaskedEdit

Comments on Each Control

One of the things you'll notice is that I use a few controls a lot, and don't make a heavy effort to use every available control. I've thought about it a lot and wonder if VB programmers out there are like me, or if everyone uses their favorites over and over? You'll have to decide for yourself which approach makes sense. In the comments that follow I provide some of the logic which pulls me to one of these controls, or what it is that keeps me away!

 **Animation Control**

Personally, I don't do games and I find animation in business programs to be of marginal
value (unless the animation is of real-time generated data). So, this is one of the controls I seldom use. Interestingly enough, I get a lot of questions from newbies on how to do animation and I steer them to this control.

 **Communications Control**

There really are some excellent reasons to become familiar with the MCI control. Programs like Laplink are hugely successful, primarily over their ability to work through the serial and parallel ports of a PC. I believe that 98% of all programmers never have that need. For the remaining two percent, however, this control can be a god-send.

 **CoolBar Control**

The concept is great - give your program a look and feel like that of other programs that your users are familiar with. In practice, most of our users have been exposed to so many windows programs that an Explorer-like interface or a Win3.1 application interface are all so easily understood that unless your user is picking between your program and a second one, I don't believe the look-and-feel will buy you any more users. Take advantage of it if you want, or just use what you're familiar with.

 **DataRepeater Control**

I tend to do things manually. Any control which automatically performs tasks for me tends to be ignored so long as I can get the job done with tools I'm already familiar with. This is a habit I really need to break. As I've said before, a programmer's job is to get the job done quickly and economically. Under those guidelines, controls such as the data repeater should be high on my list. I'll work on it!

 **DateTimePicker Control**

So far, I've never written a program which needed a calendar. I've done many where a date was entered but usually the date was the current date and that was easy to provide to the user. I can see where some programmers could put this to good use.

 **FlatScrollBar**

The only thing that would entice me to use this is that it can be made to take on both horizontal and vertical orientations. That makes it pretty versatile. Otherwise, I'm not interested.

 **ImageCombo Control**

This solves a problem which a lot of VB programmers have asked for, putting graphics in
a list. It's more effort to program than a simple text list, but if you want the graphics then it's the way to go.

 **ImageList Control**

I use the Toolbar control a lot! Since the Toolbar gets its images from this control, then

that means I have to give this one a thumbs up! I need it, I like it, and I use it all the time.

 **Internet Transfer Control**
From what I've read, this is much improved over the version that came with VB5. Many
of my users are corporate employees from engineering areas. Their needs rarely require an over-the-next solution so my experience on this one is minimal. However, since it is the **only** internet control in VB, then by definition you will use it if you have needs for Internet applications.

 **ListView Control**
The four controls - the listbox, the common dialog control, the TreeView and this one (ListView) can pretty well handle display of data in simple or heirarchical methods. It's a useful control, but not one that you cannot live without.

  **MAPI Controls**
This is an excellent example of a control which is very specialized, so much so that no one I know uses it. I have no doubt someone does, but I just haven't met them yet.

 **Masked Edit Box Control**
You'd think that this control would be one of the most popular there was and that Microsoft would have made it an intrinsic control. But, it just didn't turn out that way. The earlier versions of the control gave it a very bad reputation and it doesn't seem to have recovered from it. It's not exactly the easiest to work with and the actual operation isn't as trouble-free as I'd like. I don't use it but I think about it a lot!

 **MonthView Control**
Calendars in a program out to be very useful. Every PIM has one so why haven't my own
applications had them? Maybe if I'd had this control in VB5 I'd be further along the learning curve.

 **MSChart Control**
Yep - when I need a chart I use this one. Actually I don't like it all that much. It has a huge number of properties and I never seem to get all of them working together the way I
want. Even so, it comes with VB and it can be made to do the job, so I use it.

 **Multimedia Controll**
As I mentioned, I mostly do engineering-oriented applications. The use of .wav and .avi files would be fine, but for my needs I would want to create the .wav/.avi files myself and VB has no means of doing so. If you can use stock files from someone else, then this control is really a valuable tool. It's the only control in VB that does what it does, so it must also be the best one!

 **PictureClip Control**
Multimedia presentations and applications really are great. But as all artists know you can't just create a quality image in seconds. Having this control available to house images is fine, but where do all those images come from? Like with the Multimedia Control, this one is valuable to folks who have access to images that already exist and which meet
their own business needs.

 **ProgressBar Control**

This is the one where I am really delinquent. I owe it to my users to apply this control much more often than I do. I recommend that you use it whenever possible. The techniques of changing the cursor to an hourglass is good, but a progress bar is much better!

 **RichTextBox Control**
This is very much a specialty control. For 99% of my user inputs, a simple text input is just fine. I've gotten a lot of email from users who are using this control so I know that the lure of better looking text must be hard to resist. However, I resist it and don't really have any reason to recommend this one to you unless you are writing some kind of word processor or if you really need to display formatted text.

 **Slider Control**
Yawn. Well, maybe not. There are times when adjusting something to get a value is needed. But I could do that with the intrinsic scrollbars already. This control is easier to use but I wish Microsoft had spent their money getting a totally new feature rather than a better old one.

 **StatusBar Control**
This one is an excellent way to give your user messages about the status of the application or about the status of a user request. I don't know exactly why I don't use it more, but I'm certain that my bad attitude will change. I recommend you learn how to use this one and that you apply it to virtually every program you write.

 **SysInfo Control**
Based on questions I get from users this one should be a big winner. There are all kinds of reasons why a programmer needs to know about the PC on which his .EXE is running. This control has made a good start in providing that kind of data to an application. I'd rather have seen it built in to VB as functions but until then you can use this control to get data that otherwise would require the use of the Windows API.

 **SSTab**

 **TabStrip Control**

 **Toolbar Control**
The Common Dialog box is easily the most useful ActiveX control, and the Toolbar control takes second place. The control is reasonably easy to use and I've never had any trouble with it - it works as advertised! I highly recommend you learn to use this one.

 **TreeView Control**
Except for the lack of a built-in ability to save its content, this control has a lot of application potential. I've used it or one of its predecessors in every version of VB since VB3. Learn to use it right and it can provide a very strong user interface for the display of ordered data. If you can't figure out how to save a tree structure, email me and I'll let you in on the code I came up with.

 **UpDown Control**
Pretty simple, but actually pretty useful. It should have been intrinsic too, but it isn't. You
won't have to spend much time on it to become an expert so just learn it and then put it on the shelf until you need it. You may not need it often but when you need what it does, you'll be pleased that it is available.

 **WinSock Control**
I don't use it. However, lots of my visitor do ask questions about it and it seems to be a

pretty popular control. As soon as I bite the bullet and get more into Internet programming I suspect this one will move up the list of favorites.

Summary
As I said, I tend to use a few controls often and a lot of controls very seldom. If I was an expert in the controls I use infrequently, would I tend to use them more often? It's the old problem of not knowing what you don't know. I plan to work on stretching my control selection to give all of my less favorite controls a better chance. In a year, we'll see if my list of favorites has changed!

Code

In this section I'll cover the topic of coding - the use of the Visual Basic language. As part of this section I'll try to show you which elements of the language you need to know the most, and
which elements work together. Even more than you saw with some of the intrinsic controls, there are groups of the VB language which are almost always used together because they cover different aspects of a problem.

Overview

It's worth noting that when managers talk about programmers, one of the common metrics used
to describe performance is "lines of code per month". There's all kinds of debate about how good a metric this is, but the fact is that the metric is used!
It's not that you don't get credit for novel algorithms, or that you won't be a hero to fellow programmers when they see how you solved a problem with 10 lines of code that took them 100. You'll get that credit (and mental satisfaction, too!) but looking at the big picture it's clear that
the volume of code you can crank out will be the visible result of your efforts!
Let's walk again through the elements of writing a VB application. The percentages at the right are an estimate of how much time a programmer might spend on the various phases of the program. For short programs, a greater percentage would be spent on the concept and user interface phases. For larger programs, more time will likely be spent on the later phases. However, in either case you can see that the coding section is one of the key areas in which a programmer will be spending his time.

Concept / Requirements Definition (10%)

User Interface Design (20%)

Coding (40%)

Test & Debug (20%)

Customer Acceptance / Evaluation (10%)

Get the picture? If you want to be a great programmer, you have to know how to code!
There's another aspect to coding that you need to be aware of, but it's a little harder to explain. In the electronics manufacturing business there's a saying that 90% of the cost of a product is determined in the design. Can you see the link? Writing a program is essentially a design task, aimed at solving a problem. Like most things, there are many design approaches for every problem. Some solve the problem by grinding out an answer. Some are very elegant. Others are so complex that even the designer has a hard time keeping up with the convolutions of the approach.
Remember that in my **Training/Advice** I harped on the idea that a programmer's job was to economically provide the result for which he is being paid. A customer will be more impressed by a job done on time, on budget and which meets spec than he will be for a project with overruns because of bells and whistles (or unusual code) which do not improve the utility of the application to him or his employees.

You'll get this kind of philosophy to programming throughout this tutorial. Let's see if we can tie all this together now.

1. Great programmers must be coding experts

2. Great programmers understand that there are many approaches to a problem
3. Great programmers work efficiently. Remember my **Rule4!**

As I go through the rest of the tutorial I'll try to show how these principals can be applied to real life decisions and solutions.

Language Grouping - According to Beene

My overriding approach to learning is take what I read and try to piece it together with everything else I've learned. I think of all knowledge as one big puzzle and it's important to know where each piece of information goes in the puzzle in order to understand it's importance. In programming, knowledge is certainly a tool and as my daddy used to say, "It's the poor carpenter who blames his tool!".
The immediate value of this pearl of wisdom is shown in the following chart.

VisualBasicCommandGroupingChart

Beginner's who read through the Microsoft manuals, particularly the Language Reference manual, are often intimidated by how darn many commands they find. An alphabetical listing is simply no help in getting your arms around the problem. Back in the DOS days, several of the BASIC manuals groups commands much like what you see in the table above. This one is one that I did myself, while **reading through the entire Microsoft VB Language Reference!**
The value of such a grouping is twofold. If I had it to start with then I could have "bitten" off commands in small chunks. The small doses would consist of commands which act together to let me perform categories of tasks (file I/O, user input, loops, string manipulation, etc.). Secondly, by creating my own grouping, I forced myself to think about the code and to understand not only what the command can do, but how it can be used with other commands to get a job done!
This brings me to another philosophical point. You noticed that I mentioned having **read** the entire Language Reference? On my Beginner's page you may be noted that I also have read the Programmer's Guide front to back not once, but three times? I am a staunch believer that reading material can only be absorbed little by little. You read something once and you get 70% of it. Read it again and you get up to 90%. Third time brings you up to 95%. Better yet, write your own tutorial and you'll get some of that last percentage .
If you want to learn code, then you **have to read the manual** over and over til it makes sense to you. Each time you learn something, you're better able to learn something else. That's why rereading a manual is so valuable. The skills you pick up in the first reading make it easier to understand the parts you didn't understand the first time. I encourage you to read, read, read and while you're doing it, take notes and later summarize what you've learned. It sounds tedious, but then coding is not exactly all glitter either!
All of you have seen the acronym RTFM, right? Well the popularity of this phrase just backs up
what I've been saying. You'll get no sympathy from me for misunderstanding a topic unless you can show me where you studied the topic. Of course, it's okay to not understand the manual's explanation of a topic (Microsoft's manual are good, but they are definitely not great!) but you owe it to the person you're asking questions of to have done the leg work yourself **before** drawing on their experience to help you out.

Variables

I know you have been getting anxious to get to the **coding** part which I keep referring to, but there's one more lesson you need before we get there. The lesson is about variables. Simply, a variable is a name you use in your program which refers to a value. You've probably already seen variable names like **x**, or **i**, or **name**, or .....

You've probably already heard that values can be represented in many ways, such as integers, floating point values, strings, objects, etc.
Here's the most important list of variable types you'll run across in VB:
Integer 1, 2, 3, ... with the largest value = 32767

Long Like integers, but with the largest value about 2E9

Single 0.2, 5.7, 24.12 ... any number with a decimal

Double Like Single, but with the largest value about 2E308

Strings Not numbers at all, but letters or special characters

This isn't the whole list, but it's definitely the ones you'll use 99% of the time in your applications. I'll bring up the others as we need them in the tutorial.
This list is **not** just an exercise in useless knowledge. Later on you'll see how the use of a particular variable type can be of use in speeding up your program or of how using the wrong variable type can actually result in a wrong answer in some calculations! What you don't know can really hurt you!

File I/O

Often, books on VB don't cover this topic until much later in the book. I'm covering it early because it influences how I've grouped the VB language into the chart above.
A fundamental thing to understand is that computer files are usually stored in two basic ways, ASCII and binary.

1. **ASCII** is just an abbreviation which refers to storing information as string data. Each letter or number has a value (for example, the letter 'a' has the ASCII value 95) which you can find in VB Help by searching on ASCII. The value 12 is stored as a two character string, '1' and '2'. This takes two bytes of hard disk space.

2. **Binary** describes a way of storing data more efficiently than ASCII. The value 12 would be stored as a single byte value represented by the binary string 00001100. This approach holds for all numbers although larger numbers may take more than a byte of storage. Strings cannot be stored this way and are always stored as one byte per character.

Visual Basic has commands which can read or store in both storage methods. The storage method you use can make a big difference in how simple the code can be in your application. I won't cover all the details now, but will bring it up as needed in the tutorial. For now, here's the summary:

1. ASCII storage is most useful for storing strings. When **both** numbers and strings are stored, ASCII is normally used. VB commands for ASCII file access are usually the least complex I/O commands, but are not as flexible as binary file I/O commands.

2. Binary storage is more useful for storing numbers. VB commands let you read one or more bytes of binary data at a time and if provides more flexibility in moving around through the file.

There are, of course, other schemes for storing data. Many programs such as Word, Excel, and PowerPoint have their own storage schemes. This is also true of database program files such as those created by Access or dBase. This tutorial won't discuss proprietary formats except to note that they exist.

Printing

VB has commands which allow you to print information, either to the computer screen or to a printer. For the printer, VB creates an empty "canvas" to which your commands write, then dumps the "canvas" contents to the printer on demand or at the conclusion of a program.
For me, the single most tedious aspect of coding is that of printing reports. I **strongly advise** that whenever possible, that you make use of reporting tools such as the Crystal Reports control that comes with Visual Basic. Crystal Reports provides a very fast method of creating

printed reports against data contained within Access databases, which is native database format supported by Visual Basic.

Procedures

One fundamental aspect of VB which will become second nature to you is the idea of procedures.
When we talk about VB commands, such as "Print", or "Open", we are talking about a command
which tells VB to run a section of code which performs the Print or Open function. The code is contained in the various VB files that were installed on your machine, such as DLL files that get distributed with your applications.
With VB, like most languages, you can create your own "commands" but we call them procedures instead. The two types of procedures VB supports are called Subroutines and Functions.

1. **Functions:** These procedures are a section of code which is run when the Function is called, and a single value is returned to the calling line. For example, when you use the square root function in VB:

   x = SQR(25)
   This is a VB function called SQR. It passes the value 25 to the function and the function returns the square root and places it into x.

2. **Subroutines:** These procedures are just like a Function, but no value is returned.

It's important to note that procedure calls include a list of values (or variables) which are passed to the procedure. The procedure is allowed to change those variables when it is executed - but you make the decision by the choice of your code within the procedure.

Okay, it's coding time!

Okay, let's begin to pick apart the subject of coding. You'll remember that I described coding as something you put into an event procedure that VB will execute when the event occurs? Well, that's true but you can also add code to your application which is not contained in an event procedure, but which **must** be called from within those event procedures.
Here's how it works. If you double-click on an object in the IDE you'll be presented with the code window for that object. The upper left dropdown list shows the list of objects in your application and the upper right list shows the events that each object supports! Its an excellent way to explore the events your objects can support. If an event has had code added to it, the event list make the event title **bold**.
If you look at the top of the left dropdown list, you'll see a listing that is labeled "General". Remember those Function or Sub procedures we discussed earlier? Well you'll find all of them in this "General" section. VB automatically collects all of the user-defined function and sub procedures into this one area.
So, back to a higher level of discussion: When you enter code, you first think to put it in the events of your project objects. If you have special function or sub procedures to write (mostly because it will be used in multiple locations throughout your application) then you create a Sub or Function to put the code. Sub/Function procedures work identically except VB controls when an object event procedure is executed, whereas the code you write in the event procedures can specifically call out for your custom Sub/Functions to be executed.
By the way, you can also cause an object's event procedure to be executed at any time. For example, if you have a button (Command1), then it's click procedure is command1_click. If you type this procedure name in as a line of code, then it will execute, **just as though you have clicked the button with your mouse!**

Helpful Tip!

Visual Basic has some built in variables which can make your code very easy to read, and more
importantly, are much easier to remember than the actual value of the variable. Here's how it works:

Suppose you want to set the .CHECKED property of a checkbox to its checked state? If you look into HELP you'll see that you must see the .CHECKED property to a value of "2" for it to be checked. Easy enough? Well, consider across all the controls that there are hundres of properties

to remember. VB comes to the rescue with built-in variables. In our example, you could set the .CHECKED property to the built-in variable VBCHECKED, which VB will recognize as a "2". As you would guess, there is also a VBUNCHECKED. Both of these are visually self-explantory, whereas if you used an integer (2 or 1,in this case) you might trouble figuring our which state the code is intended to create.
VB has a lot of these easy-to-remember constants. When you read HELP for the syntax of a command/function, you will find the list of built-in constants which can be applied. **Get in the habit** of using these built-in constants to avoid the hassle of memorizing the actual values themselves!

Command Descriptions

In the earlier lesson you were introduced to coding, and reviewed a chart which groups VB commands into useful categories. In this section of the tutorial, I take each one of the commands and provide the syntax and a brief description of what the command does. For details, you can turn to the VB HELP file. My intent is to allow you to skim quickly over the commands to determine which one can help out in your situation. This page is kind of long, but I wanted to keep it all on one page to make it easier for you get scroll through it. I've also summarized the contents of this page in a **commandsummarychart**.

Operators

Here are the VB operators used to perform mathematical operations on one or more variables. Aside from the normal multiply/add/substract and divide, you will find the AND, OR, Not Equal, MOD and Integer Division operators very useful.

/ - Normal division

\ - Integer division (truncates the answer)

^ - Exponentiation operator

* - Multiply

+ - Plus

- - Minus

= - Equal

> - Greater Than

< - Less Than

<> - Not Equal

>= - Greater than or equal

<= - Less than or equal

AND - Defines a boolean value that is the AND of two values

- result = expression1 AND expression2

OR - Defines a boolean value that is the OR of two values

- result = expression1 OR expression2

XOR - Defines a boolean value that is the exclusive OR of two values

- result = expression1 XOR expression2

NOT - Defines an opposite boolean value

- A = NOT B

EQV - Performs a logical equivalence on two expressions (result is true if both expressions are true)

- result = expression1 EQV expression2

IMP - Performs a logical implication on two expressions

- result = expression1 IMP expression2

IS - Determines if 2 variables reference the same object

- result = object1 IS object2

LIKE - Determines if one string matches a pattern

- result = string LIKE pattern

MOD - Returns the integer remainder of a division

- i = 27 MOD 5

Math

VB also provides built-in functions which can act on variables. Most are self-explanatory. In my experience, the VAL, RND, and ROUND functions are among the most valuable, so be sure to pay close attention to them!

Round - Rounds a number to a selectable number of decimal places

- result = round (tempvariable,2)

Val - Returns the numerical content of a string

- result = Val ("123.4")

Int - Returns an integer by truncating (different than Fix)

- i = int (tempvariable)

Fix - Returns an integer by truncating (different than Int)

- i = fix (tempvariable)

Hex - Returns the hexadecimal value of any number

- temp$ = hex (tempvariable)

Oct - Returns the octal value of any number

- temp$ = oct (tempvariable)

Tan - Returns the tangent of an angle

- tempvariable1 = tan (tempvariable2)

Rnd - Returns a random number between 0 and 1

- tempvariable1 = rnd

Randomize - Initializes the Rnd function so it gives different answers each time

- randomize

Sgn - Returns the sign of a number

- i = sgn (tempvariable)

Sin - Returns the sine of an angle

- tempvariable1 = sin (tempvariable2)

Cos - Returns the cosine of an angle

- tempvariable2 = cos (tempvariable)

Abs - Converts a number to a positive value

- i = abs (tempvariable)

Sqr - Returns the square root of a number

- tempvariable1 = sqr (tempvariable2)

Log - Returns the base 10 logarithm of a number

- tempvariable1 = log (tempvariable2)

Atn - Returns the arctangent of an angle

- tempvariable1 = atn (tempvariable)

Partition - Sort of an oddball function but segregates values according to ranges

-

Type Conversions - A variety of conversion functions

- CBool, CByte, CCur, CDate, CDbl, CDec, CInt, CLng, CSng, CStr, CVar

Strings

In my experience these functions are used more than just about any of the other VB built-in functions. The FORMAT, MID, and INSTR functions are incredibly powerful and I use them extensively. If you don't understand what they are, they are **worth the time to figure out!** The LEN and CHR functions are also valuable as are the variations on the trim and case functions.

Left - Returns the left n characters of a string

- temp$ = left$ (teststring$, 4)

Right - Returns the right n characters of a string

- temp$ = right$ (teststring$, 4)

Trim - Removes leading and trailing spaces of a string

- temp$ = trim$ (teststring$)

LTrim - Removes only the leading spaces of a string

- temp$ = ltrim$ (teststring$)

RTrim - Removes only the trailing spaces of a string

- temp$ = rtrim$ (teststring$)

UCase - Makes all characters upper case

- temp$ = ucase$ (teststring$)

LCase - Makes all characters lower case

- temp$ = lcase$ (teststring$)

Mid - Returns n characters from a string, starting a any position

- temp$ = mid$ (teststring$, 1, 4)

Len - Returns the length of a string (how many characters it has)

- temp$ = len (teststring$)

LSet - Positions a string inside another, flush to the left

- temp$ = lset (teststring$)

RSet - Positions a string inside another, flush to the right

- temp$ = rset$ (teststring$)

Format - Returns a string formatted according to a user-defined format

- temp$ = format$ (teststring$, "####.0")

String -

- temp$ = left$ (teststring$, 4)

Chr - Returns the string representation of a number

- temp$ = str$ (32)

Asc - Returns the ASCII code of a single character

- temp$ = asc ("A")

Space - Returns n spaces

- temp$ = space$ (15)

Instr - Determines if one string is found within a second string

- i = Instr (starthere, string1, string2)

InStrRev - Determine if one string is found in a second, starting at the end

- i = InStrRev (string1, string2, start)

StrComp - Compares two strings

- result = StrComp (string1, string2)

StrConv - Converts the case of a string's characters

- StrConv (string, vbuppercase)

StrReverse - Reverses character order in a string

- StrReverse (string1)

Replace - Replaces each occurrence of a string

- Replace (bigstring, searchstring, replacementstring)

FormatCurrency - Returns a string using a currency format

- FormatCurrency(var1, 2)

FormatDateTime - Returns a date or time expression

- FormatDateTime("3/2/99",vbShortTime)

FormatNumber - Returns a number formatted according to a variety of options

- FormatNumber(var1, 2)

FormatPerCent - Returns a number formated as a percent

- FormatPerCent(var1, 2)

Arrays

Every programmer eventually uses arrays. Mostly they're pretty easy to understand. Take note, however, that you can resize an array with REDIM without losing the data. For details, see the PRESERVE keyword in the HELP entry on REDIM. If you use the LBound/UBound in your code instead of hard-coding the dimension of the array, you can later change the size of the array without touching your code!

Option Base - Determines whether the lowest range of an array is 0 or 1

- option base 1

Erase - Erases all values of an array

- erase (arrayname)

Dim - Creates an array

- dim arrayname(25)

Redim - Resets the bounds of an array (has option to save values)

- redim arrayname(28)

UBound - Returns the upper dimension of an array

- i = ubound (arrayname)

LBound - Returns the lower dimension of an array

- i = lbound (arrayname)

Filter - Returns a subset of an array based on a filter

- Filter (inputarray, searchstring)

Array - Yes, there is a function called array. It returns an array that has been filled with data from a list. It allows you to put the actual data values in the code to avoid having the user input it or to avoid having to read it from a file

- ArrayName = Array (10, 20, 30)

Join - Concatenates strings within an array

File Handling (Generic)

While VB is working on a better approach (FileSystemObject), the built-in file handling statements are still the only way to access data other than through the VB database capabilities. Your skills in this area can make or break your ability to work with various formats. The OPEN/CLOSE statements are critical to success, but the LOF, EOF, and LEN functions are used even more often! It's also a given that you'll use the DIR function regularly.

Dir - Returns a filename that matches a pattern

- temp$ = Dir ("*.*")

CurDir - Returns the current directory

- temp$ = CurDir

MkDir - Creates a directory

- mkdir ("newdirectoryname")

ChDir - Changes the current directory to a new location

- chdir ("newdirectoryname")

ChDrive - Changes the current drive

- ChDirve "A"

RmDir - Removes the indicated directory

- rmdir ("directoryname")

Freefile - Returns an unused file handle

- i = freefile

Open - Opens a file for access, locking it from other applications

- open "filename" for input as #1

Close - Closes a file so that other applications may access it

- close #1

LOF - Returns the length of a file in bytes

- i = lof (#1)

EOF - Returns a boolean value to indicate if the end of a file has been reached

- statusvariable = eof (#1)

Name As - Renames a file

- name "filename1" as "filename2"

Kill - Deletes a file

- kill "filename"

Fileattr - Returns attribute information about a file

- i = int (tempvariable)

GetAttr - Returns attributes of a file or directory

- i = GetAttr("c:\windows\temp")

SetAttr - Sets the attributes of a file

- SetAttr pathname, vbHidden

Reset - Closes all disk files opened by the OPEN statement

- Reset

FileDateTime - Returns data file was created or last edited

- FileDateTime (filename)

FileLen - Returns length of file in bytes

- FileLen (filename)

FileCopy - Copies a file to a new name

- FileCopy sourcefile, destinationfile

Lock - Controls access to a part or all of a file opened by OPEN

- Lock #1

UnLock - Restores access to a part or all of a file opended by OPEN

- UnLock #1

Width # - Set the output line width used by the OPEN statement

- Width #2, 80

File Handling - ASCII-specific

While VB is working on a better approach (FileSystemObject), the built-in file handling statements are still the only way to access data outside of a data base. Your skills in this area can make or break your ability to work with various formats. The OPEN/CLOSE statements are critical to success, but the LOF, EOF, and LEN functions are necessary to build useful code.

Line Input - Reads an entire line of ASCII text

- line input #1, tempvariable$

Write - Puts data in a file, with separators for the data

- write #1, tempvariable$

Print - Puts data in a file with no separators

- print #1, tempvariable$

Spc - Used in a print statement to move a number of spaces

- Print #2, var1; spc(15); var2

Tab - Used in a print statement to move to TAB locations

- Print #2, var1; Tab(20); var2

File Handling - Binary-specific

VB also support features which allow you to access a file on a byte-by-byte basis. The good thing about it is that you have more control, the bad thing is that you may have to write more code. Generally, a programmer will use the option (ASCII or Binary access) according to the **least** code he has to write. For binary access the Get/Put are equivalent to the Line Input and Print functions used in ASCII text file access. The big difference between the two is that binary access will read (Get) an exact number of bytes of data, and the reading can start at any byte within the file.

Get - Reads data from a file

- get #1, anyvariable

Put - Puts data into a file

- put #1, anyvariable

Seek - Moves the current pointer to a defined location in a file

- seek #1, 26

Input

- input #1, anyvariable

Loc - Returns current position with an open file

- i = Loc(#2)

Declarations

I probably get more questions about the functions in this section than about any other group. In general, the concepts are pretty simple, but the details of getting it exactly right can cause even experienced programmers trouble. Focus on understanding Dim/ReDim/Public/Private/Sub/Function/Type and Set. However, they're all useful at times, so bear down and commit these to memory. I'll try to add more text and tips on these than I have on the others.

Dim - Used to define a variable as a certain type

- i = dim i as integer, r as single
- You can use the Option Explicit to make sure that VB forces you to declare every variable you use. DIM is that simplest way to declare a variable

ReDim - Used to change the dimensions of a dynamic array

- redim arrayname(37)
- Don't be afraid of this one. You can use ReDim to create an array whose size grows by 1 every time you want to add a number to it. Then, the UBound tells you how many numbers you've added.

Static - Establishes a procedure variable which keeps its value between calls

- static i as integer

- For example, if you want to keep track of how many times you've been in a procedure, set a counter as STATIC and increment it by one for each visit to the procedure. It will never go away until the program is terminated.

Public - Creates a variable which can be accessed outside its own procedure

- public i as integer
- Even if you're the only programmer writing code in your application, use of Private vs Public will help catch errors if you inadvertently try to access an out- of-scope variable

Private - Creates a variable that can be read only in its own procedure or module, according to where the declaration took place.

- private i as integer
- Use this as often as possible to avoid unnecessary exposure of your variables to coding mistakes.

Sub - Defines a procedure which can execute a block of code

- Sub NewProcedure (var1 as integer, var2 as string)
- Be sure to check out HELP for how to handle Sub arguments. There are more questions and mistakes made concerning the use of arguments than just about anything else I've seen.

Function - Declares a procedure which can return a value

- Function NewFunction (var1 as integer, var2 as string) as SINGLE
- This is actually the most versatile of the Sub/Function procedure types. It can do anything a Sub can do as well as returning a value for use in an expression.

Call - Transfers control to a Sub or Function (is optional)

- Call Procedure 1
- Since the use of CALL is optional, forget you ever saw it

CallByName - Executes a method of an object or set/returns a property

- CallByName(form1,procedurename,vbMethod)
- The really cool thing about this is that you don't have to hardcode a procedure call. Just use a string variable with the name of the procedure to call.

Option Explicit - Instructs VB to force an explicit declaration of all variables

- Option Explicit
- You're borderline stupid if you don't use it to catch typing errors. Set up the VB IDE to automatically include this in all projects.

Option Compare - Instructs VB on how to make string comparisons

- Option Compare Binary
- This can add case-insensitivity for those times when you don't want to hard-code it

Option Private - Prevents a module's content from being referenced outside a project.

- Option Private Module
- Generally doesn't apply to most VB applications. If you find a good use for it let me know.

Property Get - Declares how to get the value of a property

- Property Get Name()
- You won't use this much until you get into creating classes of your own

Property Let - Declares how to assign a value to a property

- Property Let Name()
- You won't use this much until you get into creating classes of your own

Property Set - Declares how to set a variable reference to an object

-
- You won't use this much until you get into creating classes of your own

Set - Assigns an object reference to a variable

- Set X = form1.txtInputFromUser
- Very useful for making code more readable or simply to cut down on how much typing you have to do!

Let - Precedes assignment of a value to a variable

- Let i = 3
- It's optional, no one uses, so forget you ever saw it

Type...End Type - Creates a user defined part type which consists of standard VB data types

- type anytypename
- one as string
- two as integer
- three as boolean
- End Type
- This is a really excellent way to keep several kinds of data under one variable name. Plus, you can PUT or GET a user-defined type with a single line of code.

Const - Creates a variable whose value is fixed

- const anyname
- Basically, use this to give easy to remember names to values. For example, suppose you use the value 37.2 a lot in your code, then if you put CONST MyAge

= 37.2 in your code you'll be able to insert the MyAge where the 37.2 should have gone. Easier to type and easier to read. Also, you can chane the value of the

constant by changing only the declaration line of code, rather than searching out every place the value was used!

Declare - Used to define a procedure that exists in another file

- declare functionname (arg1 as integer, arg2 as string) as integer
-
- ArrayName = Array (10, 20, 30)
- Implements - Specifies a class to be implemented in a module
- Friend - Allows procedure to be callable from modules outside the class
- GetObject - Return a reference to an ActiveX component
- CreateObject - Creates and returns a reference to an ActiveX object
- GetAutoServerSettings - Returns information about the state of an ActiveX component's registration.
- Enum - Declares a type for an enumeration
- Event - Declares a user-defined event
- TypeName - Returns the type of data in a variable
- VarType - Returns the type of data in a variable
- DefType - Sets the default data type of variables

  DefInt A-Z
- IS - A variety of data type or status checking options

  IsArray, IsBindable, IsBroken, IsDate, IsDirty, IsEmpty, IsError, IsMissing, IsNull, IsNumber, IsObject, IsReady, IsRootFolder

Date/Time

These functions are pretty self-explanatory so I've not added any extra comments to them.

- Date - Gets the current date

- Time - Gets the current time
- Now - Gets the current date and time
- Timer - Returns the number of seconds since midnight
- DateAdd - Adds a time interval to a date
- DateDiff - Returns how many time intervals there are between two dates
- DateSerial - Returns the month/day/year
- DateValue - Returns the date
- Year - Returns the current year
- Month - Returns the current month (integer)
- MonthName - Returns the text of the name of a month
- Day - Returns the current day
- Hour - Returns the current hour
- Minute - Returns the current minute
- Second - Returns the current second
- TimeSerial - Returns a date with the hour/minute/second
- TimeValue - Returns the time
- WeekDay - Returns the current day of the week (integer)
- WeekDayName - Returns the text of a day of the week

Miscellaneous

In this list you'll find some of the features of VB about which I get a lot of email questions! The MsgBox is easily the **most used** of the bunch. It handles all of the "Y/N" queries to your user so get to know it well. Also, the **DoEvents, Shell, and Command** functions are indispensable in certain occasions so make sure you know when they should be used.

- MsgBox - A built-in dialog box that gives a message and allows a user input

```
i = msgbox "Read this!", vbokonly, "Test Message"
```

- DoEvents - Allows VB to complete pending tasks

```
doevents
```

- Shell - Executes a 2nd program from within the current program

```
shell "notepad.exe"
```

  Note - VB does not wait for the Shell'd program to quit before executing the next line of code!

- Command - Gives any text that followed a VB .EXE execution command

```
temp$ = command
```

- Environ - Returns the system environmental space content

  temp$ = environ
- Beep - Makes the computer beep once.

  beep
- InputBox - A built-in dialog box that allows entry of a text string

  inputbox "Input a value!", 5
- AddressOf - Provides an entry point for an external program to use a procedure

  AddressOf (procedurename)
- AppActivate - Activates an applications window

  AppActivate (windowtitle)
- RaiseEvent - Fires an event declared at module level

  RaiseEvent ProcedureName
- Load - Load an object

  load form1
- Unload - Unload an object

  Unload form1
- LoadPicture - Load a picture into a control property

  form1.picture = loadpicture (filename)
- SavePicture - Save a picture to a file

  SavePicture(form1.picture,filename)
- LoadResData - Load the data from a resource file

  LoadResData(index,format)
- LoadResString - Load a string from a resource file

  LoadResString(index,format)
- SendKeys - Send keys to another app as though they were from the keyboard

  Sendkeys {DOWN}
- QBColor - Returns a value corresponding to the original QB values 0-15

  form1.backcolor = QBcolor (12)
- RGB - Returns a color value by inputting the red, green, and blue parts

  form1.backcolor = RGB (12,128,256)
- Me - Refers to the current object, usually the active form

```
print Me.caption
```