Chapter 1 – What is This Thing Called VB?

# Objectives

* The History of Visual Basic?
* Advantages of VB
* Disadvantages of VB
* Understand the concepts of object-oriented and event-driven programming models

## History of BASIC

Visual Basic is both a really new and a really old programming language. By that I mean that while .NET has been around since 2002, BASIC has existed since 1964. I always think that it is worthwhile to explore how and why you are using a particular programming language.

BASIC (which stands for Beginner’s All-purpose Symbolic Instruction Code) was created by two college professors, John Kemeny and Thomas Kurtz, at Dartmouth College. Didn’t we already have programming languages back then? The answer is yes. FORTRAN (FORmula TRANslator) was considered too mathematical and thought to be best left to the scientific and engineering world, while COBOL (Common Business Oriented Language) was too verbose and was left for the business world.

Drawing from both FORTRAN and ALGOL, a new language was carved out. The thought behind it was to create a tool that would making programming accessible to everybody, especially beginners. Was it successful? You bet! For those old enough to remember the microcomputers of the 1980s, just about every one of them came with some flavor of a BASIC interpreter built in: TRS-80; Texas Instruments TI 99/4A; Timex-Sinclair 1000; Atari 400, 800, 800XL and 1040ST; Commodore VIC-20 and C-64; Apple II, IIe and IIc; & even the original IBM-PC.

BASIC was a great language for beginners. Anybody could learn to program in it. One of the big problems that BASIC suffered was that unstructured program code ***could*** be created in it. Care could be taken to avoid writing bad code though. The truth of the matter is that any programming language can produce junk code. However, BASIC really got bashed for it probably because so many people were playing around in it, learning how to program and writing poorly structured code as a result. Another issue that forced BASIC to be viewed as a “toy” language was the fact that it was interpreted and not compiled. On the generation of computers available in the 1980s, interpreted code was too slow to be considered for serious systems development – therefore “real” developers would never use it.

BASIC eventually started to die out as a popular programming language as the microcomputer market started to contract. In addition, languages such as Pascal and C were taught at the college level to computer science students. Companies such as Microsoft and Borland released fairly cheap compilers for these languages causing the use of interpreted BASIC to decrease even further.

Then came a major shift from Microsoft… Many people believe that Microsoft only recently entered into the BASIC arena with its Visual Basic product line. This really isn’t true at all – in fact Microsoft’s first product was a BASIC interpreter created for the old Altair computer system, which was several years before the DOS days of Microsoft. Bill Gates has a love for the BASIC language – he wrote part of the Altair interpreter; this is often evidenced by the fact that new advancements in technology showed up in the Visual Basic language first before being added to the other Microsoft languages.

Many people learned to program in BASIC on the IBM-PC since it was the language that accompanied the DOS operating system. Originally the language was called MS-BASIC on the IBM-PC. Clone machines had the same version but the name was changed into GW-BASIC. The GW stood for Gee-Whiz in case you were wondering.

Around the late 1980s, Microsoft’s chief competitor in compilers, Borland, created a product called Turbo Basic. This was a major breakthrough in resurrecting the BASIC language since Turbo Basic was compiled and eliminated the need for line numbers in the language. Essentially this product was among the first to show that BASIC could be every bit as structured as languages like Pascal and could produce tight, fast object code through compilation.

Microsoft then followed suit by creating QuickBasic. QuickBasic was basically (no pun intended) Microsoft’s version of Turbo Basic. The QuickBasic compiler was fairly successful, enough so that Microsoft started bundling a smaller subset of Quick Basic with DOS 5.0. This product was called QBasic and replaced the old GW-BASIC that had been bundled with DOS.

The original creators of BASIC decided to get back into the act in the late 80s. Since BASIC was enjoying a resurgence in popularity, they decided it was time to try to get a piece of the pie. Another version of BASIC, called True Basic, was released that was touted as being a more “pure” version of BASIC than the other dialects that were available, notably QuickBasic and Turbo Basic.

Alan Cooper, in the early 1990s, realized that the process of developing Windows applications was both lengthy and difficult. To simplify the process, he developed a GUI prototyping application system called Tripod that allowed the used of dragging and dropping Windows controls to construct graphical interfaces. Microsoft contracted Cooper’s company to add a programming language to the system. The code that would glue everything together would be a modified version of QuickBasic (this project was called Ruby – no relation to Ruby on Rails). In 1991, Microsoft welcomed Visual Basic version 1.0.

It didn’t take Microsoft long to realize the power that VB offered to developers. The idea was to implement reusable components and then glue them together with a little bit of code to form applications – this was one of the premier rapid application development environments. Creating a Windows based graphical interface was now trivial – the VB language was friendly and easy to use. As such Microsoft continued to add power to the VB environment. It was always far easier to develop software using VB than in any other Windows programming language. VB 4 was the first version that really started adding OOP constructs to the language. The biggest drawback to version 4 is that it was half compiled/half interpreted as p-code (bytecode).

Visual Basic 5, in addition to providing an incredible programming environment, created true native compiled programs that executed every bit as fast as Visual C++ programs. The other major limitation of VB 4, its inability to create reusable controls natively within VB, was also eliminated in VB 6. VB 6 also added additional object-oriented constructs and the ability to create ActiveX Document projects that executed through a web browser. VB 6 was a solid development environment that didn’t take up a lot of hard drive space nor require lots of system resources.

Enter VB.NET in 2002 – Microsoft’s newest take on a development suite. The architecture behind .NET requires all languages to generate what is essentially bytecode (can anyone say Java?). Microsoft’s vision is to allow developers to use whatever languages they want and to even mix them, if they so choose. In order to pull this off, Microsoft had to make all languages internally the same. VB.NET is part of Visual Studio .NET – you can’t just load VB anymore without having the .NET framework. The .NET framework is coming preinstalled with modern versions of the Windows operating system. Today’s Visual Studio IDE, in contrast to the small VB6 IDE, takes 20 to 130 GB to install depending on what’s selected (ouch!).

# Advantages of Visual Basic

Now that we’ve taken a tour of where Visual Basic comes from, let’s take a look at the relative advantages of Visual Basic:

* Visual Basic comes in three flavors: Visual Basic, Visual Basic for Applications (VBA) and VBScript. If you learn one, you can pretty much program in all three. Visual Basic can be used to create complete applications from scratch. Visual Basic for Applications can be used to extend the functionality of applications that have VBA built in (like Office). Finally, VBScript can be used to create both client-side and server-side web scripts. Microsoft’s Active Server Page.NET (ASP.NET) technology allows you to use VBScript and VB as a language.
* There is nothing that today’s VB.NET cannot do compared to other languages in the .NET framework such as C#.
* VB promotes reusability through components and object-oriented classes.
* VB forces you to learn some object-oriented programming skills. VB.NET is completely object-oriented just like Java. Even though most of the language is procedural, you must use some OO development techniques to write VB.NET code: it’s just not avoidable!
* VB will allow you to create standalone programs, web programs and custom controls all from within the same familiar environment.
* VBA runs behind the scenes in many applications like Microsoft Office, Visio and AutoCAD. If there is something that these applications don’t do for you, you can extend the application through custom VBA code.
* VB is, by far, the easiest Windows application development system you will come across! No case sensitivity, semicolon or brace nuttiness!

## 

## Disadvantages of Visual Basic

Hopefully by now I’ve convinced you that VB is the *only* way to develop. You should be asking the question of “what’s wrong with VB?” Here’s a list of some of the various issues that people complain about:

* It is a Microsoft product and not a standard, nor is Visual Studio a free application. If you want to use open software standards, VB is probably not the way to go. While it is widely developed in, VB is owned and controlled by Microsoft. Microsoft has recently open sourced the VB Roslyn compiler and there is the standardized CLR runtime. Microsoft has, over the past several years, tried to be a good citizen in the open source community and has extended its vision through .NET Core to other platforms (Linux, Android, iOS). Only time will tell how widely and well received this is…
* For the most part .NET is only available on the Windows platform. The Mono project sponsored by Novell has made a decent chunk of .NET available for Linux developers, but, it’s always playing catch up because Microsoft isn’t slowing down. Finally, the OpenOffice product allows you to program in a language that looks suspiciously like VBA. Microsoft has started evangelizing about moving its .NET platform to other operating systems beyond Windows, so maybe this “negative” will change down the road.
* .NET uses a VM-type architecture. This means that true executable files are not produced, instead an intermediate bytecode language is generated which is compiled to native object code as needed, just-in-time. This is somewhat funny since Microsoft told Sun back in the day that Java wouldn’t live due to that reason – “compiled code will always be superior.” You will notice that applications start out slower until the first instantiation of each object. Execution speed will increase after the initial instantiations. Over the generations of .NET, execution speed has gotten better and better to the point that in some cases it now equals native code execution. Let’s face it: hardware keeps getting faster and faster, so VM architectures aren’t going to be going anywhere.
* .NET and VB 6 (and older versions of VB) are incompatible in many ways. This means that it is not a trivial task to get VB 6 programs to run under .NET. There are a lot of these still out in industry in use! While in the long run the common language runtime will prove to be a good feature, it is quite disheartening to see how many VB 6 applications cannot be directly upgraded to .NET without programmer intervention, and sometimes only with a lot of programmer intervention.

# Miscellaneous Terminology

As with anything new, there are two terms that you need to understand. If you’ve been programming for a while, these concepts are probably quite familiar to you. If you’re new to the GUI development world, then you need to make sure you understand what these terms mean and how they differ.

*Object-Oriented*: The idea of trying to model real world objects on the computer by examining and reproducing the attributes (data) and methods (actions) of the real world object. For example, a car has the attributes of a color, an engine size, a stereo, number of doors and so forth. The car has the methods of starting, accelerating, braking, opening the door, turning a key and so on. Objects communicate with one another through the use of message passing. We create object “cookie cutters” through the use of a class. Inheritance, Aggregation, Polymorphism, Interfaces, Overloading, Overriding – these are all object-oriented concepts.

*Event-Oriented*: Event-oriented systems mean that software must respond to events that are generated by various actions which occur. For example, moving the mouse, pressing a key on the keyboard or having a timer “tick” all generate events that a program must deal with. To further explore this, how many ways can you print a document in most programs? You can use the File🡪Print menu option; you could click on the Printer button on the button bar/ribbon; or you could press some keyboard hotkey combination like ALT-P. All three actions generate the same event – the fact that you want to print a document. Your program would then implement an event handler that is called to respond to an event when it is raised. The event handlers for all three of the print actions listed would call the same print document function. Another interesting idea is that we never know when an event will/might be raised – this gets us into having to worry about asynchronous behavior in our programs.

Visual Basic uses both object-orientation and event-orientation to create solutions for your Windows based programming problems. By no means is VB exclusive in these concepts; almost all modern programming languages that support the development of a graphical user interface or support the use of devices such as mice use both objects and events.

# The Bottom Line

I still strongly recommend Visual Basic as the best tool to develop Windows platform software with. The speed and capability of what you can do with VB still has no rival. Should you learn VB.NET? Absolutely! Not only will it give you the ability to create top notch Windows applications, it will conceptually prepare you for migrating to other languages such as Java which also uses a large framework architecture.

A word of caution: watch Microsoft somewhat carefully – the company doesn’t always tend to solve the deep computer science problems that exist; instead they sometimes market eye-candy. Microsoft created its new C# language to try to lure Java programmers to .NET. They created and then killed J#, which still isn’t Java, but looks a lot more like it. C# already looked like Java!!!! Why did they create another language? We have F# now. Tools and technologies come and go (Silverlight, a competitor to Adobe Air, is out; the slick Object Workbench tool disappeared from Visual Studio). Bottom line – you need to be careful where you invest your time in learning and applying Microsoft technologies. Of course, this isn’t just a Microsoft problem: look at Java’s deprecated feature list and passé technologies like Swing.

If you know that you are going to develop on multiple platforms, I strongly recommend you spend some time researching the Java SE platform. Until .NET becomes available on multiple platforms, this is the only language that exists which will run on Windows, UNIX, Linux and Apple as a “native” application.

Now that we’ve taken a road trip through the history of and what’s good/bad about VB.NET, let’s start exploring what’s available and how to use it.