The truth about P-Code

Introduction

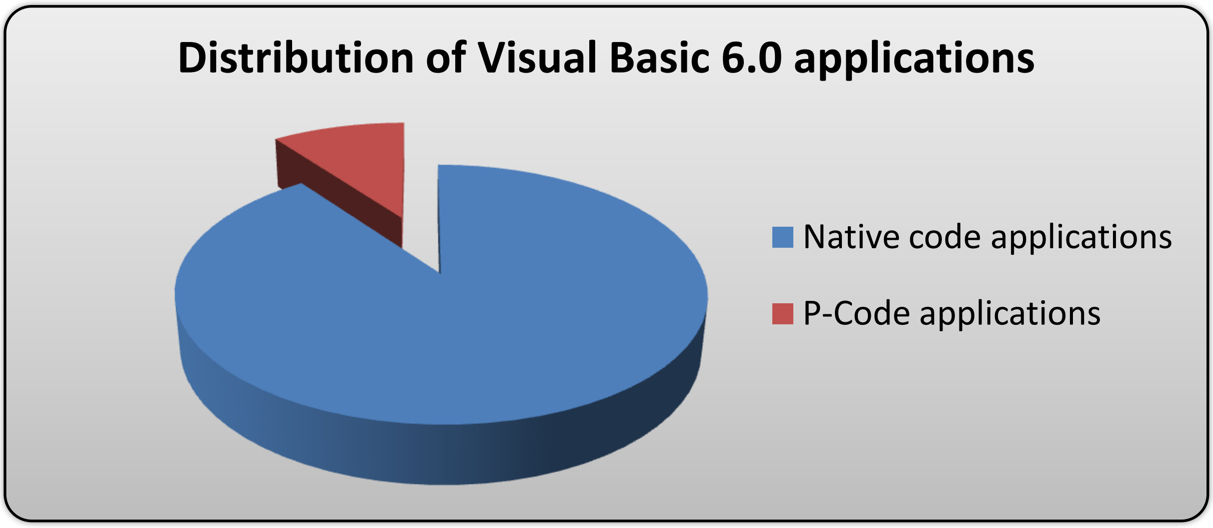
This article has been written to provide better and indispensable information to users of decompilers since I receive many complaints from customers of others Visual Basic decompilers because their application weren't recovered by these decompilers.

The reason is, when you write a Visual Basic 6.0 application you may choose a P-Code or Native code compilation, then your application will be compiled to P-Code or Native code which are very different approaches of compilation, so the decompilation will be different too.

There is a question you may absolutely ask yourself when you need a decompiler for a specific application: *Is my application compiled into native or P-Code mode ?*

The main problem is that a lot of users got a P-Code decompiler in order to decompile their application, but released with Native Code…

In fact, P-Code decompiler are today almost useless because 90% of Visual Basic 6 application are released with Native Code mode. This article will explain first the difference between P-Code and Native code, and then explain why the rate of Native code application is so important compared to P-Code applications.



I. P-Code Versus Native Code

When you write a line of code in the IDE, Visual Basic breaks it down into expressions and encodes the expressions into a preliminary format called op-codes. In other words, each line is partially precompiled as it is written. Some lines contain shared information that cannot be precompiled independently (mainly Dim statements and procedure definitions). This is why you have to restart if you change certain lines in break mode. The opcodes are compiled into p-code instructions when you compile (in the background if you have the Compile On Demand and Background Compile options set).

At run time, the p-code interpreter works through the program, decoding and executing p-code instructions. These p-code instructions are smaller than equivalent native code instructions, thus dramatically reducing the size of the executable program. But the system must load the p-code interpreter into memory in addition to the code, and it must decode each instruction.

It’s a different story with native code. You start with the same opcodes, but instead of translating to p-code instructions, the compiler translates to native instructions. Because you’re not going to be expecting an instant response while stepping through native code instructions in the IDE, the compiler can look at code from a greater distance; it can analyze blocks of code and find ways to eliminate inefficiency and duplication. The compiler philosophy is that, since you compile only once, you can take as long as you want to analyze as much code as necessary to generate the best results possible.

These two approaches create a disjunction. How can you guarantee that such different ways of analyzing code will generate the same results? Well, you can’t. In fact, if you look at the Advanced Optimizations dialog box (available from the Compile tab of the Project Properties dialog box) you’ll see a warning: "Enabling the following optimizations might prevent correct execution of your program." This might sound like an admission of failure, but welcome to the real world of compilers. Users of other compiled languages understand that optimization is a bonus. If it works, great. If not, turn it off.

On the other hand, very few developers are going to be used to the idea of working in an interpreter during development but releasing compiled code. Most compilers have a debug mode for fast compiles and a release mode for fast code. Visual Basic doesn’t worry about fast compiles because it has a no-compile mode that is faster than the fastest compiler. You get the best of both worlds, but it's going to take a little while for people to really trust the compiler to generate code that they can't easily see and debug.

Source : http://vb.mvps.org/hardcore/html/p-codeversusnativecode.htm

II. Proportion of P-Code application in the world

The distribution of Visual Basic P-Code application is very small compared to Visual Basic Native application (90% of Visual Basic 6 applications are compiled with Native Code setting -default setting in VB6-), that is one of the reason why I decided to base VBReFormer on Native Code decompilation instead of P-Code decompilation.

The great spread of Visual Basic Native application compared to P-Code applications can be explained by the fact that the default compilation option in Visual Basic 6.0 is set to « Native Code » compilation, of course because native application execution time is almost as fast than in C++ applications contrary to P-Code applications which add an execution level.

Before choosing a decompiler you must know if it was released for Native Code application or P-Code applications, and if your application was released in P-Code or Native mode.

Note that P-Code is easier to decompile than Native Code because of its high level properties.

To know if your application is compiled into Native or P-Code, you will need help. For that purpose I made a little application which will tell you if your applications were compiled into Native or P-Code (http://www.decompiler-vb.net/vbcomptester_en.exe).

Sylvain Bruyere
Decompiler-VB.net Developper
Microsoft Student Partner
http://www.decompiler-vb.net/