CHAPTER 4

The Syntax for the QuickBasic Compiler

The use of Cobol cripples the mind, and its teaching should be regarded as a criminal offense.
—Edsger Dijkstra

It is practically impossible to teach good programming style to students that have had prior exposure to Basic; as potential programmers, they are mentally mutilated beyond all recognition.
—Edsger Dijkstra (in a foul mood)

The late, hero computer scientist was just wrong about Basic. Dijkstra’s comment is academic sociology at its worst. It creates the illusion that programming skill derives from the use of politically correct platforms and languages. Dijkstra was wrong because Visual Basic is Turing-complete, and it has a formal and sensible syntax. Visual Basic is Turing-complete because you can use it to write any program, as long as you disregard resource consumption.

I would revise Professor Dijkstra’s aphorism. The use of Basic or Cobol as representative of a good programming language in and of itself cripples the mind and rots the teeth because Cobol and Visual Basic preserved (until Visual Basic .NET) some standards and practices created in the Fortran era, which simply did not allow for effective problem breakdown. This was less a scientific fact and more a result of a management illusion that programmers should merely code specifications provided by the “real” experts, and not factor the problem into subroutines, functions, and objects. Much sloppy programming results from this false view of the field and the low self-esteem it creates in programmers, who believe that a disreputable language permits mindless coding. In actuality, the

1. In an interview, Peter Neumann, long the hard-working moderator of the comp.risks newsgroup, told me that Dijkstra struggled with depression most of his life. Many bright people are depressed because they are powerless to stop other people from making mistakes. Dijkstra, unlike many successful corporate MIS types, never restrained himself from speaking his mind. His attempts at constructive criticism sometimes bothered people who had heavily invested in a paradigm Dijkstra did not like. Paradoxically, all who knew Dijkstra personally said he was easy to get along with.

E. G. Nilges, Build Your Own .NET Language and Compiler
© Edward G. Nilges 2004
reverse is true: we should compensate for the deficiencies of the language by mindful coding.

In the 1970s and after Dijkstra made these comments, Basic compiler developers added structured constructs to the language, and commencing with Visual Basic 4, Microsoft has been adding object-oriented tools.

Dijkstra said and wrote a lot of things, all of which are thought-provoking, but not all of his views have stood the test of time. One strange aphorism that still holds is that “computing science is no more about computers than astronomy is about telescopes.”

Dijkstra meant that it’s a mistake to focus on tools rather than the job at hand. Don’t forget that the computer and the programming language are your telescope, and the stars are the user's problem and your solutions. The programmer’s job is to bring the stars down to earth.

Basic has a reputation as being vague in syntax and not formalizable, as are more recent languages like Java and C++. As you’ll see, this isn’t so. Underneath its clunky, wordy, and keyword-intensive syntax, Basic can be completely and formally specified using Backus-Naur Form (BNF), and a tool for analyzing BNF can be written in Visual Basic itself. This brings the stars down to earth, as long as we cease fussing about the deficiencies of the telescope, stop hacking at it, and grow up and get to work. I will also show you how to convert BNF to Extensible Markup Language (XML), which gives another sensible view of BNF syntax.

In this chapter, I will discuss the rules for coding BNF by actually applying an analyzer, the bnfAnalyzer program, to the syntax for BNF itself. Then I will describe the construction of the BNF for our version of QuickBasic. We will then run the syntax of our QuickBasic through an analyzer and examine the output. We need to verify that it is analyzed without error and forms a solid basis for the compiler we will start to build in Chapter 5. As a summary, we'll look at eight guidelines for effectively developing BNF. This chapter will conclude with a special section on bnfAnalyzer internals, best read after you’ve read Chapters 5 through 8.

BNF is a valuable tool for specifying sensible .NET languages. Don’t skip this chapter. If you propose to design a language for business rules, text processing, or making home movies, always create a solid BNF as your detailed requirements analysis.

A Tool for Analyzing BNF

In this chapter, we’ll use a program called bnfAnalyzer to load and analyze the syntax of QuickBasic, expressed in BNF. The bnfAnalyzer executable program is available from the Downloads section of the Apress Web site (http://www.apress.com). You’ll find the code in the egnsf/bnfAnalyzer folder.