CREOLE FORTH FOR VB(A)

JOSEPH M. O'CONNOR

FORTH 2020

JUNE 2021

WHY BUILD A FORTH FOR VB6

- In this case, because someone asked for it.
- In January of this year, Peter Forth put me in touch with Buzz Ross.
- Buzz has a background in Forth for embedded systems but has also done a lot of development work using Visual Basic.
- He was interested in combining the strengths of both languages.
- This short presentation will summarize my work on building a version of Forth for VB6.
- It is similar in structure to other versions of Forth I have bult for my own use.

DESIGN

- Partitioned into the following VB class files:
- AppSpec.cls – reserved for application-specific primitives
- ArrayList.cls – used as the basis for the stacks
- CompileInfo.cls – container used to help build a high-level definition
- Compiler.cls – colon compiler, defining and compiling definitions
- CorePrims.cls – code for core primitives such as DUP, SWAP, ROT, -ROT, and basic math
- CreoleForthBundle.cls – builds the table of primitives and high-level definitions
- CreoleWord.cls – defines a Creole Forth word
- GlobalSimpleProps.cls – generic object that all primitives takes as a parameter. Has the stacks, other globally-available data structures, push, pop.
- Interpreter.cls – outer interpreter, DoColon and vocabulary words
- LogicOps.cls – logical operatives
- LoopInfo.cls – container for DO..LOOP iteration
- ReturnLoc.cls – container for return stack information

CREATING WORDS

- All words are based on the CreoleWord object
- They come in two forms:
- - Primitives. Write code in VB, then reference with the BuildPrimitive method.
- - High-level definitions. Generally built with the colon compiler. CREATE/DOES> pair is available too. They make use of the parameter field while primitives generally do not.

EXAMPLE

- In CorePrims.cls module:
- ' (n1 n2 -- sum) Adds two numbers on the stack
- Function DoPlus(ByRef poGSP As GlobalSimpleProps)
- Dim dblVal1 As Double
- Dim dblVal2 As Double
- Dim dblSum As Double
- Call poGSP.Pop(poGSP.DataStack)
- dblVal1 = CDbl(poGSP.Scratch)
- Call poGSP.Pop(poGSP.DataStack)
- dblVal2 = CDbl(poGSP.Scratch)
- dblSum = dblVal1 + dblVal2
- poGSP.Scratch = dblSum
- Call poGSP.Push(poGSP.DataStack)
-
- DoPlus = 0
- End Function

EXAMPLE 2

- After setting up in CorePrims module, add a dictionary entry in CreoleForthBundle.cls with the BuildPrimitive method:
- Call BuildPrimitive("+", "CorePrims", "DoPlus", "FORTH", "COMPINPF", "(n1 n2 -- sum) Adds two numbers on the stack")

HIGH-LEVEL DEFINITIONS

- Call BuildHighLevel(": 3H 3 0 DO HELLO LOOP ;", "Three hellos")

THE COLON COMPILER

- Has no state variable
- Immediate words are all in the IMMEDIATE vocabulary
- Compilation begins when IMMEDIATE is pushed onto the vocabulary stack.
- This means IMMEDIATE words are always first in the search order during compilation.
- When compilation is terminated, the IMMEDIATE vocabulary is popped off the vocabulary stack. This prevents this vocabulary from being accessible when not compiling.

IS IT USABLE IN MICROSOFT OFFICE?

- Yes, just Import the listed cls files and make sure Microsoft Scripting Runtime is referenced in Tools→References. There is an example of this in my Github repository.
- If you use it with Excel, primitives like **+. -, *, /, <, >, =, <=, >=** don't react well when placed at the beginning of Excel cells.
- Because of this it may be advisable to give them substitute names such as N+, N-, N*, N/, LT, GT, EQ, LE, GE, etc.

PROOF OF CONCEPT - DEMO APPLICATION IN EXCEL

- Steps to building:
- 1. Import the class files
- 2. Change the names of the primitives +, -, *, / % to N+, N-, N*, N/, N% to be more compatible with using an Excel cell as a ‘command line’.
- 3. Do the same with the logic primitives =, <>, <, >, <=, >= become EQ, NE, LT, GT, LE, >GE.
- Write an interfacing macro or macros.
- Write any other necessary primitives or high-level definitions. Application-specific primitives can go in the AppSpec class.

QUESTIONS?

- Code is available on Github at http://github.com/tiluser/cfvb