# HOW TO BUILD PRIMITIVES IN CREOLE FORTH FOR EXCEL

BY JOSEPH M. O'CONNOR

# QUESTION: HOW DO YOU WRITE PROGRAMS IN CFE?

- Short answer: you don’t.
- Instead, you write an instruction set to solve your problem(s).
- Two ways to write these instructions.
- A.  As primitives. These are methods of a class that consist of raw VBA code and are set up using the BuildPrimitive subroutine.
- B.  As high-level (colon) definitions. Defined with : (colon) and terminated with a semicolon.

- This tutorial will concentrate on writing new primitives.

# MORE ON PRIMITIVES

- Other Forth compilers often allow the programmer to write in assembly language with the CODE/END-CODE pair.
- It may be helpful to think of CFE primitives as analogous to writing in its “assembly language”.

# HOW TO BUILD A PRIMITIVE, THE SHORT VERSION

- 1. Write the function in a class module. It must take GlobalSimpleProps as an argument and return 0.
- 2.  Set up an entry in the dictionary with a BuildPrimitive call.

# STEPS TO WRITING A NEW PRIMITIVE, PART 1

- 1. Open CFExcel1.xls. Make sure macros are enabled.
- 2. Alt-F11 to write new code.
- 3. Navigate to the TutorialCode class module.
- 4.  Find the code for the function DoHelloFromCFE.
- 5. Copy the code and its comment immediately above.
- 6. Paste in the code to the AppSpec module.

# STEPS TO WRITING A NEW PRIMITIVE, PART 2

- 7. Copy the line of DoHelloFromCFE  with the function name and its comment immediately above.
- 8. Navigate to the AppSpecMain module.
- 9. Find the GenBuildPrimDef2 subroutine and place your cursor inside it.*
- 10. Hit the F5 key. A dialog box will pop up with the generated code for the BuildPrimitive call.
- 11. Cut and paste this code into the BuildDefs sub.
- 12. Change the “XXX” to “HELLOCFE”. This will be the name the word will be called by.
- 13.  Type Ctrl-R and Ctrl-C to rebuild the dictionary and global data structures sheet.

# TESTING THE PRIMITIVE

- 1. Make sure you've rebuilt the dictionary and the primitive is listed in it (check the Dictionary tab).
- 2.  Type "HELLOCFE" without the quotes in the Input Area.
- 3. Hit the Submit button.
- 4.  A dialog box saying "Hello from CFE" should come up.

# IF YOU WANT TO DO MORE…

- The previous example is admittedly very simple.
- There are no stack effects, for one thing.
- One of the great things about Forth is its easy parameter passing via the stack.
- So the next example will be a primitive that puts some data onto the stack.

# A RANDOM NUMBER GENERATOR

1. Copy the function DoRand100ToStack in the TutorialCode module to AppSpec.
2. In AppSpec, copy the function name and its comment immediately above.
3. Go to the AppSpecMain module.
4. Navigate to the body of the GenBuildPrimDef2 sub and hit the F5 key.
5. Cut and paste the generated code into the BuildDefs sub.
6. Change “XXX” to “R100>TOS”.
7. Rebuild with Ctrl-R Ctrl-C and type R100>TOS into the Input Area.
8. A random number between 1 and 100 should appear on the stack.

# RULES/GUIDELINES FOR SETTING UP NEW PRIMITIVES

- 1. Set up a function in a class module that takes a GlobalSimpleProps object and returns 0.
- 2. Create an entry in the dictionary with a call to the BuildPrimitive sub.
- 3. The simplest way to do this is to use the AppSpec class module and the AppSpecMain regular module.
- 4. Declare all variables before using them, since Option Explicit is defined.

# STACK HANDLING

- 1. Pushing data
- poGSP.Scratch.Value = iRandValue
- iReturnVal = poGSP.Push(poGSP.DataStack, poGSP.DataStackMarker)
- 2. Popping data
- iReturnVal = poGSP.Pop(poGSP.DataStack, poGSP.DataStackMarker)
- sMsg = poGSP.Scratch.Value

# QUICK DEVELOPMENT TIPS

- If there's a bug in your primitive, remember to press the 'Empty Stack' button. It will clear the Data Stack, Return Stack, and Parsed Input.  Not doing so can leave the environment in an invalid state.
- Sticking a breakpoint in the primitive and single-stepping through is a great way to figure out any problems or what's going on "under the hood".
- While it's possible to define primitives outside of AppSpec/AppSpecMain, it requires more setup work in the CreoleForthMain module.  For small programs, it should be sufficient.
- The other class modules (CorePrims, Interpreter, Compiler, etc) should be reserved for the base language and not be used for application-specific primitives.  This is an organizational guideline though, not a hard-and-fast rule.