COWS Interpreter

Driving System

COWS Version 1.3

Sean Luke

March 20, 1994

This file describes, in English, what the various COWS Interpreter functions are doing in Objective-C. It is not entirely accurate, but it should give you a general idea of what's going on.

The interpreter works by starting up a pulsing function in GO, and letting the

system run to completion. The interpreter maintains its own internal stack (see COWSStack.h), onto which it places a number of different nodes, just like many things are stuck on a real machine's stack. The stack is theoretically infinite in size.

You don't need to understand the internal workings of the interpreter to connect your application to it, or to write a COWS program! It's just here for those daring few who might want to make their own interpreters or modify mine and want to understand how the heck this thing works.

Some Caveats

I assume that errors automatically return without finishing out a method.

This system is started up by calling EXECUTE PROGRAM, which executes some initial function in the program defined by Symbol Name, using the

arguments in Argument List.

The main function in the COWS Interpreter Driving System is READEVAL, which is driven by a timed entry (clock) or by a pulsing function through GO. If functions drop out, it's assumed that they just finish, return control to the timed entry or pulsing function, and READEVAL starts up again. In some cases I've marked a function as "continuing to ReadEval". This is the same thing...I just got sick of writing it...

Because this interpreter is pulsed, it is **not** reentrant! Reentrance should be simulated through pausing the interpreter, initializing another, calling a function on that interpreter, destroying it, and resuming the interpeter with the return value. Oh, and if that weren't all, this won't work if the interpreter is synchronous.

Interpreter Functions

EXECUTE PROGRAM: Argument List, Symbol Name

Set Interpreting format to Undefined
Push Symbol Name
Push Arguments from Argument List, so last argument is on top
PERFORM FUNCTION
it is the responsibility of the calling program to free its argument list and symbol name afterword

PERFORM FUNCTION

Pop Arguments into Argument List
Pop Symbol Name
If Symbol Name is in Function Dictionary
    PERFORM INTERNAL FUNCTION:Argument List, Symbol Name
Else if Symbol Name is in Library Dictionary

PERFORM LIBRARY FUNCTION:Argument List, Symbol Name
Else
Error: No Such Function
Destroy Argument List

PERFORM INTERNAL FUNCTION: Argument List, Symbol Name
If Interpreting format is Undefined, set it to Internal-Function
Save current state (if there is none, ignore it...)
Push local state based on Symbol Name
<Set current position, etc. based on new local state...>
Grab argument names, associate with argument list values in local argument dictionary.
If not enough arguments,
Error: Not enough arguments...
Grab variable names, add to local variable dictionary.
Move to beginning of function, and continue to READ EVAL

PERFORM LIBRARY FUNCTION: Argument List, Symbol Name

If Interpreting format is Undefined, set it to Library-Function
Look up target/selector by Symbol Name
Call target with selector and argument list
If Interpreter was paused by target
    Continue to READ EVAL *Though that won't do much...*
If return value is an error,
    Error: (Some Library Function Error)
Else
    Push return value
    Continue to READ EVAL

DO KEYWORDS

Get top symbol
If top symbol is If
    DO IF
If top symbol is Set

DO SET
If top symbol is For
DO FOR
If top symbol is While
DO WHILE
Else
Continue to READ EVAL

COMPLETE FUNCTION

If top item is a value
Pop it and set the Return Value to it
If not
Set the Return Value to ""
Pop off everything to and including the Local State (if there is oneÐthere might not be if the user just asked for a library function)
Get New Local State (above a Library Function Node)
If New Local State exists,

Set Current Function to New Local State's function string
Set Current Position to New Local State's position
Set Current Dictionary to New Local State's Dictionary
Push Return Value
DO KEYWORDS
If not (i.e., stack is empty)
program is finished...
Stop Interpreting

EVALUATE VARIABLE: Symbol Name
Get Local State
If Symbol Name is in Local State Dictionary
Push value
If Symbol Name is in Global Dictionary
Push value
If in neither dictionary
Error: No such variable

GO *main clock function. Calls READ EVAL*

While running

call READ EVAL <repeats> times

note that this could be done either through a timed entry or through multiple pulsing until it's done or someone pressed Command-period.

READ EVAL

Get next token

If there's no token

If Interpreting format is Library Function

COMPLETE FUNCTION (user just asked for a library function-- obviously, there'd be no tokens)

Else

Error: Out of Tokens

If token is string or number or truth

```
    Push it
    DO KEYWORDS
If token is a symbol
    EVALUATE VARIABLE:Symbol Name
If token is (
    Get next token
    If there's no token
        Error:   Out of Tokens
    If next token is not a symbol,
        Error:   No Function or Keyword
    If next token is If
        START IF
    If next token is Set
        START SET
    If next token is For
        START FOR
    If next token is While
```

```
        START WHILE
    If next token is a symbol
        Push it
If token is )
    Get top symbol above the top dictionary
    If there is no such top symbol
        COMPLETE FUNCTION
    If top symbol is If
        FINISH IF
    If top symbol is Set
        FINISH SET
    If top symbol is For
        FINISH FOR
    If top symbol is While
        FINISH WHILE
    If top symbol is some other symbol
        PERFORM FUNCTION
```

START SET

Get next token
If next token is a symbol
    Push Set Node
    Set Set Node to Start_Set
    Load Set Node with token
Else
    Error: Can't Set: not a variable

DO SET

If Set Node is Start_Set
    Set Set Node to Done_Set
Else
    Error: Can't Set: too many values

FINISH SET

```
    If Set Node is Done_Set
        Pop Value
        Pop                         Pops Set Node
        Push Value
        If Set Node's Variable is in Local State Dictionary
            Set Variable in Dictionary to Value
        If it's in the Global Dictionary
            Set Variable in Dictionary to Value
        If it's in no dictionary
            Error:   Can't Set:   no such variable
    Else
        Error:   Can't Set:   No value
```

START IF

```
    Push If Node
    Set If Node to Start_If
```

DO IF

If If Node is Start_If
Pop Value
If Value is "t"
Set If Node to Start_Then
If Value is not "t"
Set If Node to Start_Else
Skip an Item *skips an item, or nothing if just hits a)*
If Pos is the same,
Error: Can't complete If: Not enough values
Note this is because if *must have* then *but not necessarily* else
If If Node is Start_Then
Set If Node to Done_If
Skip an Item
If If Node is Start_Else
Set If Node to Done_If

```
    If If Node is Done_If
        Error:   Can't complete If:   too many values

FINISH IF
    If Node is Start_If or Start_Then
        Error:   Can't complete If:   Not enough values
    If Node is Start_Else or Done_If
        Pop into Return Value
        Pop to If Node
        Pop                 Pops If Node
        Push Return Value


START WHILE
    Push While Node
    Set While Node's Pos to Pos
    Set While Node to Start_While
```

DO WHILE

```
If Node is Start_While
    Pop Test
    If Test is T
        Set Node to True_While
    If Test is F
        Set Node to False_While
        Skip
    Push Test
If Node is True_While
    Set Node to Done_While
If Node is Done_While or Node is False_While
    Error:   Can't Complete While:   Too many values
```

FINISH WHILE

```
If Node is True_While or Node is Done_While
```

Pop to While Node
Reset Pos to While Node's Pos
Set Node to Start_While
If Node is False_While
Pop into Return Value
Pop to While Node
Pop *Pop While Node*
Push Return Value
If Node is Start_While
Error: Can't Complete While: Not Enough Values

START FOR

Get next token
If next token is a symbol
Push For Node
Set For Node to Start_For
Load For Node with token

```
    Else
        Error:   Can't For:   no variable
```

DO FOR

```
    If Node is Start_For
        Pop StartValue
        If StartValue is a number
            Load StartValue into For's Start
            Set Node to End_For
        Else
            Error:   Can't For:   Start Value isn't a number
    If Node is End_For
        Pop EndValue
        If EndValue is a number
            Load EndValue into For's End
            Set Node to Step_For
```

```
    Else
        Error:   Can't For:   End Value isn't a number
If Node is Step_For
    Pop StepValue
    If StepValue is a number
        Load StepValue into For's Step
        Set Node to Do_For
        Look up Variable
        If Variable   is in Local or Global Dictionary
            Set Variable to StartValue
            Load Pos into For's Pos
            If Variable beyond End limits
                Set Node to False_For
                Skip
        Else
            Error:   Can't For:   no such variable
    Else
```

Error: Can't For: Step Value isn't a number

If Node is Do_For
    Set Node to True_For

If Node is False_For or True_For or Test_For
    Error: Can't For: Too many values

FINISH FOR

If Node is True_For
    Pop into
    Reset Pos to For's Pos
    Set Node to Test_For

If Node is False_For
    Pop into Return Value
    Pop *Pops For Node*
    Push Return Value

If Node is Test_For
    Increment

```
    If Test is finished
        Pop into Return Value
        Pop                    Pops For Node
        Push Return Value
    Else
        Pop into
        Reset Pos to For's Pos
        Set Node to Test_For
If Node is Do_For or Step_For or Start_For or End_For
    Error:   Can't For:   Not enough values
```