COWS Interpreter

Tour

COWS Version 1.3

Sean Luke

March 20, 1994

The COWS interpreter is *not* one of the smallest or easiest-to-understand objects around. This guided tour introduces you to all of the interpreter's functions, and should give you an idea of how the interpreter works, where it fits in the scheme of things, and generally how to connect it to your application.

Step One: Creating an Interpreter

Create an interpreter in the standard way: ⇒ **[[COWSInterpreter alloc] init];**

Step Two: Loading Libraries

After initializing an interpreter, an application needs to plug function libraries into it. In particular, it would be wise (but not mandatory) to plug the Standard Library into the interpreter. All this plugging-in is accomplished through the **addLibrary:** method, which instructs the interpreter to send load a compliant library.

You can also load a single library function by calling **addLibraryFunction:::**, passing it the selector and target of the method to call, and the name the function is to be referred to in COWS programs. Of course, this library function must adhere to standard library and library function protocols.

Note that if two library functions have the same name, the last one to load will wipe out the earlier one as far as the interpreter is concerned.

If you want the interpreter to communicate with and control your application, you need to give it a function library of your own making. Currently this must be done programmatically, building your own library. It's not too tough: take a look at the Jungle Gym library as an example (*COWSExampleLibrary.h* and *.m*) All libraries, including your own, are expected to adhere to the *InterpreterToLibrary* protocol declared in *COWSProtocols.h.*

Step Three: Setting Options

The COWS interpreter informs its *delegate* that it has finished some COWS program, or that it experienced an error while interpreting the program. You can set the interpreter's delegate through **setDelegate:**. To give the interpreter no delegate, you can set the delegate to NULL (the interpreter starts this way). You can find the current delegate through **delegate**. Delegates are expected to adhere to the *InterpreterToAppDelegate* protocol declared in *COWSProtocols.h.*

The interpreter can run in two modes: *foreground* or *background* mode. These modes are set using **setForeground:**, and you can find the interpreter's current mode through **foreground.**

In *foreground mode*, the interpreter takes control of the application. While the interpreter is running, you cannot do anything to the application. This is

fast and safe for interpreter programs (you can't do anything to the application that would mess up assumptions your program is making), but it denies you control. The only way to stop an interpreter in foreground mode is by pressing *Command-period.* And if some library function takes its time, you may not be able to cancel the interpreter at all.

In *background mode*, the interpreter competes with you for control of your application. This way you retain control over the interpreter (you can even cancel it by pressing the *stop* button), but all bets are off if you move around windows or open documents, etc., while the interpreter is running. COWS programs make a lot of assumptions about the current state of an application, and since you share control, you can change that state. Be careful!

So it's your pick: safe for you (background) or safe for your interpreted program (foreground).

The interpreter works by calling a special method repeatedly, pulsing the

interpreter onward to its eventual goal. This method then performs *repeats* number of interpreter "interpretations". You can set *repeats* with the **setRepeats:** method (query it using **repeats**). If you set *repeats* low (1 is the lowest), the interpreter will work very very slowly but be very easy to cancel. If you set *repeats* high, the interpreter will work quickly but be tough to cancel. You decide.

If the interpreter is working in foreground mode, it calls this special method constantly. But if it's running in background mode, the method is called periodically from a timed entry. You can set the speed of this pulsing through the **setTimedEntrySpeed:** method (query it using **timedEntrySpeed**). NeXT claims timed entry speeds are in seconds per timed entry, but they're really notÐI've found 'em to be about 1/3 of what you'd expect. A typical timed entry speed would be 0.1. Don't set the speed to 0 or a negative numbr.

Lastly, the interpreter can be *locked* to keep outside agents from accessing it.

For example, the IPC library allows one application's interpreter to control another app's interpreter from afar, even from another machine. But if your interpreter is locked, it ignores any outside requests for control. You query locked status through **locked** and set it through **setLocked.**

Step Four: Loading a Program

Your app now feeds an interpreter a program, written by the user or pre-provided by the app. This is done with the **setProgram:** method. Upon receiving this program, the interpreter parses through the program, cutting it into global variables (which it hashes into the global dictionary) and functions (which it hashes in the function dictionary). Functions are further parsed for arguments and local variables, placed in subdictionaries, and removed from the function text.

All this makes COWS faster when actually interpreting a function, since it doesn't have to search through the program to find any function or variable definition.

Step Five: Interpreting a Function

An application starts a COWS function running by calling the **interpretFunction: arguments:** method. The app should then just sit back and wait until the interpreter's delegate receives the message that the interpreter is finished. At any time, the interpreter can be stopped with the **stopInterpreting** method.

The **interpretFunction: arguments:** returns the interpreter itself. So how do you get a return value from an interpreted program? Through the delegate. The interpreter calls your delegate's **finishedInterpreting:::**

method when it has an answer for you. Currently *thisMessage* is unused. If there is an error, the interpreter calls the **ErrorInterpreting:::::** method. *thisError* is one of the errors defined in *CowsInterpreter.h.* *thisString* is usually the token in error, unless the error is COWSLIBRARYFUNCTIONERROR, in which case *thisString* is an error message from the library function in question.

Step Six: Freeing or Clearing an Interpreter

You can clear an interpreter's functions through the **clearLibraryFunctions** method. You can clear out a program by loading a new one with **setCOWSProgram:**. You free an interpreter with **free.**

Debugging an Interpreter

Since they're hash tables, it's ordinarily not easy to debug the interpreter's dictionaries at any given time using gdb. You can print out the interpreter's dictionaries and program through the **printDictionaries** and **printProgram** methods.

Pausing an Interpreter through a Library Function

The interpreter can *not* be paused by the user or the application, but it *can* be paused by function libraries that need a lot of time to return an answer or do work, if and *only* if the interpreter is running in asynchronous mode. Libraries pause an interpreter through the **pauseInterpreting:** method, and resume interpreting with the **resumeInterpretingWithValue:** method, returning the library function's return value.

Again, *do not use these methods to allow the user or application to pause the interpreter!*

Querying Interpreter Status

When the interpreter is working on a function, it's *working* flag is set, which can be queried through **working.** When the interpreter is currently running (it's not paused as described above, and it's *working*), it's *running* flag is set, which can be queried through **running.**

Private Methods (A Peek)

There are a zillion private methods in the interpreter. All start with "_". All are declared in COWSInterpreter.h. Call *none of them,* on pain of death!

It's dangerous to do so. But to give you an idea of how the interpreter works:

The Tokenizer for the interpreter is **_tokenize:::.** It eats through a program given in *string*, starting at postion *pos*, and places the next token in *token,* returning the position in the string just after the token.

Recursive Descent Parser Functions and Terminal Functions are entered using **_program::.** The job of these functions is to parse through your COWS program and break it into functions, global variables, etc., putting each in a dictionary. These functions constitute a formal recursive descent parser (for those who care).

Interpreter Functions are entered using **_executeProgram::,** and begin pulsing through the interpreted program using **_go.** The job of these functions is to do the actual work of interpreting your program once it has been parsed into dictionaries.

Interpreter Keyword functions interpret the keywords *set, if, for,* and *while.*

Interpreter Assistance Functions are assorted helper functions that skip through tokens, mold badly-formed truths and numbers and strings into proper ones, and report errors.

For more secrets on how the interpreter does its work, see *COWS Interpreter Driving System.rtf*. The recursive descent grammar (slightly different from the grammar given in *COWS Language Formal Specification.rtf*) is given in *COWS Interpreter Parsing Grammar.rtf.*