COWS

Protocols

COWS Version 1.3

Sean Luke

March 20, 1994

One of the key parts to the COWS interpreter is the COWS protocols package, which is still in a state of flux, but I think fairly robust. These protocols describe how interpreters communicate with libraries, with applications, and even with interpreters in other applications.

Controlling an Interpreter from an Application

The protocol *LibraryControl* describes the standard suite of methods a COWS-compliant interpreter should have to be controllable by an application when the application wants to load libraries, or when a library wants to temporarily stop the interpreter:

```
- addLibrary:this_library;
```

*Adds a compliant library. This tells the interperter to call the library's **loadLibrary:sender** method, which in turn repeatedly calls...*

```
- addLibraryFunction:(const char*) this_name
    selector: (SEL) this_selector
    target: this_target;
```

Adds a compliant library method to be registered as the function name

this_name. *See the next section about requirements of what these methods must look like.*

`- clearLibraryFunctions;` *Clears all library functions.*

`- pauseInterpreting:calling_library;`
Pauses the interpreter. This is used only by libraries that need to release control to the user, or whatnot, before returning an answer, must only be used in background-mode interpreters, and when the interpreter is ready to return

an answer, this method must be followed with....

```
- resumeInterpretingWithValue:(COWSStringNode*)this_value;
```

*Starts up the interpreter again, returning a value to be pushed on the interpreter's stack. This is **only** to be used in conjunction with **pauseInterpreting:** above.*

Communicating with Libraries

The protocol *InterpreterToLibrary* describes the methods interpreters may call on libraries:

`- loadLibrary:sender;` *Instructs a library to start loading functions into the interpreter using*

addLibraryFunction:selector:target

as described above.

`- pauseCancelled:sender;` *Tells a library that it's requested pause, as described above, was cancelled by the user by stopping the interpeter. After receiving this method, a library shouldn't call* ***resumeInterpretingWithValue:****.*

In addition, each library function registered by the library is expected to comply with the following format:

- *functionName*: arg_list;

Where arg_list is always a COWSArgumentList of arguments the user's passing to the function. The method is free to rip into the argument list (any nodes it pops it must free), but must not free the argument list object itself. The method then should return a fresh COWSStringNode containing the function's return value. The interpreter will free this node. If the function needs to report an error, it can do so by placing an error message in this node, and setting the node to error using **setError:** (see *COWSStringNode.h).*

Informing a Delegate

When an interpreter has finished interpreting, it returns its answer to the application (or to a remote calling interpreter) through an assigned delegate. *InterpreterToAppDelegate* describes the methods this delegate must have:

```
- finishedInterpreting:
    (const char*)returnValue:(int)thisMessage:sender;
```

Tells the delegate that the interpreter finished interpreting, and that its return value is returnValue. thisMessage *is currently unused.*

```
- errorInterpreting:(int) thisError:(const char*)thisFunction:
    (int)thisPosition:(const char*)thisString:sender;
```

Tells the delegate that an error occured interpreting. thisError *is the error (error codes are in* COWSInterpreter.h). *thisFunction is the function body in which the error occured.* thisPosition *is the position in the function where the error occured.*

thisString *is usually the token in error, unless the error was a library error, in which case* thisString *is an error message from the library.*

General Interpreter Control

The protocol *InterpreterControl* describes methods an application can use to control an interpreter in general.

```
- (int) setProgram:(const char*) this_string;
```

Loads a user-specified program into the interpreter and breaks it up. The interpreter returns an error code (see COWSInterpreter.h) *if there's an error in parsing through the*

program initially. Even if the program loads properly, it may still have syntax errors reported later.

```
- setDelegate:this_delegate;
- delegate;
```

Sets and queries for the delegate.

```
- interpretFunction:(const char*) this_name
    arguments:(COWSArgumentList*)these_arguments;
```

Interprets a function, passing it arguments. This function must actually exist in the program or in some library, and the program must already have been loaded with ***setProgram:***

above.

This is how users start COWS macros.

`- stopInterpreting;`

Cancels the interpreter's current work.

This could be connected to a big red CANCEL button in the application which the user can press...

Inter-Process Communication

The protocol *InterpreterIPC* is used by the COWS IPC Library to communicate to other COWS IPC libraries and ultimately to other interpreters in other applications. It is not actually used by interpreters themselves.

```
- sendFunction:(const char*) this_name:reply;
```

Called from a remote application which

wants your interpreter to interpret the function this_name *synchronously,*

placing the return value in reply. *This method should return NULL.*

```
- (oneway void) sendOutFunction:(const char*) this_name;
```

Called from a remote application which

wants your interpreter to interpret the function this_name *asynchronously, ignoring any return value.*

```
- addArgument:(const char*)this_argument;
```

Called from a remote application to begin loading arguments for an up-and-coming request (one of the two above).

Arguments are added first-to-last. This method should return NULL.

```
- (BOOL) isForeground;
```

This should return whether or not your interpreter is running in a foreground (work-to-the-end) or background (event-driven) mode.