COWS

IPC Library

COWS Version 1.3

Sean Luke

March 20, 1994

The COWS IPC (Inter-Process Communication) Library provides some elementary functions to allow a COWS interpreter in one application control the *interpreter* of another application. It is not designed to control applications that don't have a COWS interpreter built-in.

The IPC Library is not finished, and it's still buggy, but I've decided to get some of it out as fast as possible to give you a taste of what COWS can really do.

Distributed Objects and the Library

The library uses NeXT's Distributed Objects system to communicate with applications on your local machine or on remote machines. When an application starts up, its interpreter registers itself with the Network Name Server under the name *appname*(COWS), where *appname* is the app's name

as found through **[NXApp appName].** Currently, if two apps have the same name, only one is seen by the library.

In a non-threaded application (i.e., most NeXTSTEP apps), the Distributed Objects system must work through events. This means that there are some of things you *can't* do with an interpreter like COWS:

- You can't have your application call itself through this library.
- If app 1 calls app 2 synchronously, app 2 should not call app 1 until it returns an answer.

Synchronous vs. Asynchronous Calls

You can use the library to make both synchronous and asynchronous calls to a remote application. A *synchronous* call is a call that waits for an answer back from the remote application before continuing. An *asynchronous* call returns immediately without checking to see if the remote application has completed the requested task.

In this version of the library, only foreground interpreters can accept synchronous calls, though both foreground and background interpreters can send synchronous calls. Anyone can send or receive asynchronous calls without problems.

Local vs. Remote Calls

This library can make either *local* calls to applications on your computer, or *remote* calls to applications on other computers. Remember that if you're making remote calls, you must have an application running on the remote machine that you are allowed to talk to.

Security

The library provides very minor security at the moment. Currently, you can set an application to be *locked.* This prevents other interpreters from accessing it. However, if your interpreter is unlocked, *any* interpreter can access it from *any* machine, given the owner of the interpreter knows the name of your machine and the process running on it. You've been warned!

What This Library Can't Do (Yet)

Currently this library can't:

- Check to see if an application exists.
- Control applications without interpreters (that's really the job of a later, different library).
- Allow synchronous calls to non-foreground interpreters.
- Protect your interpreter from some remote interpreters while allowing other interpreters to control it. It's all-or-nothing.

IPC Function Names

The moment you've been waiting for...

launch Tries to launch app *app-name,* possibly on the remote machine *machine-name.* Returns *t* if succeeded, *f* otherwise.

(launch *app-name***)**
(launch *app-name machine-name***)**

launched? Returns *t* if *app-name* is running locally, or running on the remote machine *machine-name* if this is included as an argument. Returns *f* if *app-name* is not running.

(launched? *app-name***)**
(launched? *app-name machine-name***)**

send Synchronously calls a COWS function *function-name* with arguments *arguments* in the interpreter in the application *app-name*. Waits for *function-name* to complete its task. Returns what *function-name* returns. For example,

⇒ **(send "SomeApp" "doit" 13 4 "hello")**

calls the function *doit* in the interpreter in the application *SomeApp.app,* passing it the arguments *13, 4,* and *hello.* If *doit* returns a 5, *send* will return a 5.

(send *app-name function-name arguments**)**

send-remote Identical to *send,* except that it calls a function on an

application on the remote computer *computer-name.* *computer-name* is a registered domain name.

(send-remote *computer-name app-name function-name arguments**)**

send-out Asynchronously calls a COWS function *function-name* with arguments *arguments* in the interpreter in the application *app-name*. Returns nothing. For example,

⇒ **(send-out "SomeApp" "doit" 13 4 "hello")**

calls the function *doit* in the interpreter in the application *SomeApp.app,* passing it the arguments *13, 4,* and *hello.* Without waiting for *doit* to complete, *send-out* immediately finishes and returns a blank string as its return-value, assuming that *doit* is still out there somewhere doing its work.

(send-out *app-name function-name arguments**)**

send-out-remote Identical to *send-out,* except that it calls a function on an application on the remote computer *computer-name.* *computer-name* is a registered domain name.

(send-out-remote *computer-name app-name function-name arguments**)**