XEROX

INFORMATION PRODUCTS GROUP

Systems Development Division

July 19, 1977 6:33 PM

To: Pup and FTP Package Users

From: Hal Murray, x4539

Subject: How to get at the Pup and FTP Packages

Stored: <Murray>PupFTP.Bravo

Everything needed to interface to the Pup Package in defined in PupDefs. The FTP package interface is defined in FTPDefs. Both packages also reference MesaDefs, and SysDefs. If you are looking for some simple examples, the Mesa PupTest package (probably in <Murray>Pt.DM) is probably a good place to start. The NameLookup module (in the Pup Package) also has a nice example of how to use the Socket level interface. Snarf.mesa is a simple FTP example.

The Pup Package and optionally the FTP Package comes preloaded as an appendage to the Mesa runtime routines. This makes a fat image file if all the symbols are included for debugging, but you can discard your Mesa.image. A Statistics package, and a window manager modified to use our secheduler is also available as an option. I will brew up an image file with the desired options when you need it. You can also make a disk that has all of the sources and xm's, and use it to build your own image file. Everything just barely fits on one disk, but you will probably have to delete the Compiler when you are finally ready to make your image file.

You can't BIND to any of the pup routines because we have fixed up the binding path to hide everything except a tiny interface module. This makes binding to the system faster. The interface module is called Coolie - be sure to get Coolie.xm if you are getting an image file without symbols. The actual routines in the Pup Package and the FTP Package are referenced via two giant dispatch vectors. There is a third vector that the Pup Package and the FTP Package use to interface to the Mesa Runtime routines. It avoids binding, and can readily be used by other programs. I will add things to the giant mesa vector if there is something you need that isn't already there. The vectors live in file segments so they don't even use any core when not being used.

There are routines (on the binding path) that return pointers to the needed dispatch records. Normally, a manager would pass these pointers to other modules at NEW time where they would be OPENed on the module BEGIN.

```
GetMesaVector: PROCEDURE RETURNS [MesaDefs.MesaFacilitiesHandle];
GetStatsVector: PROCEDURE RETURNS [SysDefs.StatsInterface];
GetPupVector: PROCEDURE RETURNS [PupDefs.PupInterface];
GetFTPVector: PROCEDURE RETURNS [FTPDefs.FTPFacilitiesHandle];
```

NB: the mesa slots in the Pup vector, and the Mesa and the PupInterface slot in the FTP vector don't get setup until PupPackageMake or FTPPackageMake is called, so don't use either too early if you have called GetPupVector or GetFTPVector - call GetMesaVector if you need the mesa vector.

Normally, the Pup Package includes a statistics gathering section and some debugging aids. They can be supressed (by setting doStats in SysDefs to FALSE and recompiling the world) if core space is more important. GetStatsVector and GetFTPVector will generate some obnoxious ERROR if their option hasn't been loaded up. A WindowManager that has been modified to use our scheduler also comes as an option in case you need one. (RunUser, an FTP user interface program needs it.)

```
PupPackageMake: PROCEDURE RETURNS [PupInterface];
PupPackageDestroy: PROCEDURE;
```

PupPackageMake increments a use counter, and if it was zero, calls GetPupVector and GetMesaVector, builds some internal tables, allocates the pool of free packet buffers, locks some code into core, and turns on the Ethernet hardware interface. (A copy of PupPackageMake lives in a place where the binder will find it.) PupPackageMake must be called before any other routines (except GetPupVector) in the Pup Package. PupPackageDestroy decrements the use counter, an if it goes to zero, it undoes everything that PupPackageMake did, returning all the core that the Pup Package allocates. It does not destroy any sockets, PktStreams or ByteStreams left dangling. **NB:** The pointer returned by PupPackageMake is not valid after PupPackageDestroy has been called - beware of dangling references. If you need a pointer that will remain valid, call GetPupVector.

```
FTPPackageMake: PROCEDURE RETURNS [FTPInterface];
FTPPackageDestroy: PROCEDURE;
```

FTPPackageMake increments a use counter, and if it was zero, calls PupPackageMake and builds some internal tables. (A copy of FTPPackageMake lives in a place where the binder will find it.) FTPPackageMake must be called before any other routines in the FTP Package. FTPPackageDestroy decrements the use counter, an if it goes to zero, it undoes everything that FTPPackageMake did, returning all the core that the FTP Package allocates. It does not destroy any listeners, servers, or users left dangling. **NB:** The pointer returned

by FTPPackageMake is not valid after FTPPackageDestroy has been called - beware of dangling references. If you need a pointer that will remain valid, call GetFTPVector.

There are two ways to get at either the Pup Package or the FTP Package. One: get a pointer to the interface vector, probably by calling GetPupVector during your initialization code, and keep the vector locked in core. In this case, a manager could pass the vector to other modules at NEW time where they would be OPENed on the module BEGIN. Then, the worker programs would be written just like they had opened a defs file. If the Pup Package is on all the time, this doesn't cost any core. Two: don't lock the vector until you need it. In this case, you have to be sure that all copies of the pointer are updated if it has been passed around. This mode is appropiate for programs that are trying to conserve core, and only reference the Pup Package or FTP Package from a single module, and run most of the time with the Pup Package turned off.

There is one more interesting routine on the BINDing path.

```
SetSandBarOK: PROCEDURE [BOOLEAN];
```

When the file segment used to store the giant PupVector is brought into core, it might wind up in the middle of the only remaining "large" block that is left. To avoid fragmenting core, the Pup Package normally flushes all the code before bringing in the PupVector. If you don't like the disk rattling, and won't be bothered by core fragmentation, call SetSandBarOK[TRUE] to suppress swapping out all of the code modules. The same flag and mechanisim are used when locking the Ethernet driver code in core (by PupPackageMake) and for the FTPVector.

SCHEDULER

The Pup Package runs under a **nonpreemptive scheduler.** If you don't have one, we supply a nice simple one. The only primitives needed are ScheduleeCreate and ScheduleeYields. (Destroy is implemented by returning to the scheduler.) If you want to supply your own scheduler, just store your own procedures into the ScheduleeCreate and ScheduleeYields slots into the pup vector before calling PupPackageMake. When the Pup Package needs to wait for an external event it calls ScheduleeYields. There are several hidden processes needed by the Pup Package. They must be run every now and then, so user programs should also yield artifically during extend periods of computing. Things should be ok if all the non pup processes taken as a group run for 1/2 second or so.

Packets will be lost if interrupts are disabled for too long. Currently, code swapping and scrolling the display (BITBLT) are both done with interrupts disabled. At full speed, about 14 packets are lost while a normal sized mesa display gets scrolled up one line. Except for performance considerations, this shouldn't make any noticable effect. Both PktStream and ByteStream modes will recover by retransmiting.

NB: Don't call any of the get routines from two different processes for the same connection at the same time. It probably doesn't make sense, and we don't know what will happen. Crashes and/or screwups are possible. Similarly, don't try to put to one connection from two different processes at the same time. One process getting, and another putting is ok. Also, beware that some other process isn't doing a get or put when you destroy a connection.

If you are going to use the keyboard, you will probably want to call SetIdleProc[ScheduleeYields] so that the scheduler will continue to run other processes while one is waiting for the user to kit a key. You will probably have to qualify ScheduleeYields to get it out of the PupVector. If you need to get at ScheduleeYields from a program that uses the FTP Package, but don't otherwise need the PupVector, PupInterface in the FTPVector can be used after calling FTPPackageMake.

OTHER GOODIES IN PupInterface

There are several other variables in the PupInterface record that may be of interest to users.

```
myHost: INTEGER;
myNetwork: INTEGER;
```

This is simply the address of the local machine. Note that myNetwork will be zero if there arn't any gateways up, or the system is running in the local mode.

```
bufferPoolSize: INTEGER,   -- default is 10
stormy: BOOLEAN;
showit: BOOLEAN;
localOnly: BOOLEAN;
```

bufferPoolSize is the number of buffers to allocate when PupPackageMake is called. The default of 10 is reasonable for one active connection, or a few if you don't mind occasional interference. It should work with as few as 5 buffers if you don't mind poor performance. stormy controls a debugging aid. If TRUE, lightning will strike various packets, and they won't get delivered. Normally, this is uninteresting, but if you are accessing the Pup Package at the socket level, this will allow you to debug retransmission procedures on a single machine. showit activates a routine that prints the header of every packet. Its very verbose and slow, but sometimes helps to find out what is really happening. (It probably won't work in some of the special shunts.) localOnly is a debugging aid to make things reproducible. If is it set before calling PupPackageMake, then the Ethernet interface won't be turned on. This prevents gateway packets from introducing any perturbations, so core clobbering bugs can be tracked down systematically. (It also prevents you from communicating with any other machines, but you can talk to other processes on the same machine.)