Inter-Office Memorandum

To	Communication Protocols	Date	July 1, 1978
From	Ed Taft	Location	Palo Alto
Subject	The Pup Network Directory and the PUPNM JSYS	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<Pup>PupDirectory.bravo

This is a revision of a memo of the same title dated June 27, 1975. Only minor changes have been made. The principal motivation for this revision is to re-issue the memo in Bravo and Press formats.

In a separate memo, *Naming and Addressing Conventions for Pup*, we proposed the establishment of a *network directory* associating names and addresses of entities such as networks, hosts, and processes at Parc. This memo documents the format of the directory that has been implemented, the procedures for maintaining it, and a Tenex JSYS called PUPNM for performing name/address conversions using it.

The design that has evolved has benefitted from contributions by Bob Metcalfe, Dave Boggs, and Peter Deutsch.

Contents of the Network Directory

Each entry in the directory consists of a list of *names*, a list of *addresses*, and a list of *attributes*.

A *name* is a string consisting of alphanumerics plus the characters '-' and '/' (others may be admitted by popular demand). Both upper and lower case alphabetics may be used; two names differing only in the case of their letters are considered identical. If more than one name appears in an entry's name list, all the names are synonyms and may be used interchangeably.

An *address* is a triple consisting of a *network number*, a *host number*, and a *socket number*, as defined in the *Pup specifications*. In the case that all three of these numbers are nonzero, the address completely specifies a Pup *port*. However, if one or more of the numbers is zero (i.e. unspecified), the address represents a subset of all possible ports.

The classes of addresses that we expect to find useful in practice are the following:

1. Network addresses, in which only the network number is specified.
2. Host addresses, in which the network and host numbers are specified.
3. Port addresses, in which all three numbers are specified.
4. Well-known sockets, addresses in which only the socket number is specified.

The addresses in a directory entry's address list describe alternative ways of accessing the entity associated with the entry (or multiple, identical instances of that entity). Accordingly, we require

that all addresses in an address list have corresponding patterns of specified and unspecified elements.

An *attribute* is merely a pair consisting of an attribute name and an attribute value. The attribute name is composed according to the same rules as the names in a directory entry's name list, while the attribute value may be an arbitrary string.

In addition to the attribute list associated with an entire directory entry, individual addresses in the entry's address list may have attribute lists associated with them.

Structure of the Network Directory

The network directory is currently maintained on Maxc1 as the highest-numbered version of the file <System>Pup-Network.Directory. It is distributed automatically by the PUPSRV program to all other Name Lookup servers, using the Pup Network Directory Update protocol, described in a separate memo.

The file is constructed in a manner intended to facilitate interpretation by Altos and Novas. It is written in 16-bit bytes (packed in the standard PDP-10 format, i.e. two bytes left-justified in each 36-bit word). All *pointers* in this file are 16 bits wide and refer to 16-bit bytes relative to the start of the file. All *strings* are BCPL-style, i.e. an 8-bit byte count followed by that number of 8-bit bytes. As a concession to Maxc, all *blocks* and *tables* start at Maxc word boundaries, i.e. pointers to them are always even. Unused bytes are always zero.

We now present the format of the various blocks and tables in the directory, in the order in which they actually appear in the file. The individual items in these objects are 16-bit bytes except where noted.

Header Block

0	Number of name blocks
1	Pointer to name lookup table
2	Number of address blocks
3	Pointer to address lookup table
4	Number of words occupied by entry blocks
5	Pointer to first entry block
6	Version number of this file

Name Lookup Table

(Ordered by name using the ASCII collating sequence, except that lower-case letters collate with the corresponding capital letters)

Pointer to name block
Pointer to name block
...
Pointer to name block

Address Lookup Table

(Ordered by value of network, host, and socket numbers)

Pointer to address block
Pointer to address block
...

Pointer to address block

Entry Block

(One per network directory entry)

Pointer to first name block in list
Pointer to first address block in list
Number of attributes
Pointer to first attribute name block
Pointer to first attribute value block
...
Pointer to last attribute name block
Pointer to last attribute value block

Name Block

Pointer to next name in entry's list (zero marks end)
Pointer to owning entry block
Name string

Address Block

Pointer to next address in entry's list (zero marks end)
Pointer to owning entry block
Network (8 bits), Host (8 bits)
Socket (32 bits)
Number of attributes
Pointer to first attribute name block
Pointer to first attribute value block
...
Pointer to last attribute name block
Pointer to last attribute value block

Attribute Block

Attribute name or value string

Syntax of Port Names

A *port name expression* is composed of name strings and address constants joined by the operator '+'.

A *name* is one of the name strings defined in the network directory. Its value is the associated list of addresses.

An *address constant* is in the form

<network number> # <host number> # <socket number>

where the numbers are specified in octal. An element of this constant may be left unspecified by supplying zero or by leaving it out entirely. Leading '#'s may be omitted. For example, "0#0#3", "##3", and "3" all denote an address constant with network and host numbers unspecified and socket number 3, while "3##" denotes network number 3.

Names and address constants may be combined by means of the '+' operator, which is roughly speaking an intersection operator. Its effect is to make an expression whose value is more specific (i.e. contains fewer unspecified elements) than either of its operands. For example, the value of "3##+##123" is "3##123". If a particular element is specified in both operands but with conflicting values, the intersection is empty.

When either of the operands is a name whose value is a list of addresses, the resulting value is also potentially a list. For example, the value of "Maxc1" in the network directory is the list 1#1#, 2#1#, 3#200#, 4#40#. Hence the value of the expression "Maxc1+123" is the list 1#1#123, 2#1#123, 3#200#123, 4#40#123. However, the value of the expression "3##+Maxc1" (or "Parc-Net3+Maxc1") is a single address, 3#200#, since the intersections of the given address constant with the other addresses in the list are empty.

The PUPNM JSYS

To permit easy conversion between names and addresses in Tenex, we have implemented a rather elaborate JSYS called PUPNM. We shall first summarize its calling sequence before covering some of its more grandiose features.

Accepts in

1: Source/destination designator (must be string pointer if source).

2: B0 off: Convert address pointed to by RH 2 to name string on destination designated by 1 (unless B4 on; see below).

B0 on: Lookup name string designated by 1, return corresponding address(es) in table pointed to by RH 2.

B1: For B0 off: output a string in the complete form "network+host+socket" if B1 off; omit fields where possible if B1 on. For B0 on: Allow recognition if B1 on.

B2 (only if B0 off): If off, give error if address not found in network directory; if on, output an address constant using octal numbers.

B3 (only if B0 off): Return network directory address block pointer in 3.

B4 Lookup attribute name string designated by 4 and output the corresponding attribute value string to 1 (B0 must be off, and B4 on suppresses outputting of the name string to 1 and forces B2 off).

B9-17: Address table length (words, ignored unless B0 on).

RH: Location of table in which addresses are passed or returned.

4: String pointer to attribute name (if B4 on).

PUPNM

Returns

+1: Unsuccessful, error # in 1.

+2: Successful:

1: Updated where relevant (string pointer).

2: Updated only if B0 on in call:

LH: Number of words returned in address table.

RH: Unchanged

3: Only if B3 of 2 on in call:

Zero if the address passed to PUPNM did not exactly match some address in the network directory. If a match was found:

LH: Version number of network directory.

RH: Index in file (16-bit bytes) of first word of the matching address block.

4: Updated where relevant.

Addresses are passed or returned in a table pointed to by RH 2. Each address is stored in two words, with the network and host numbers in the left and right halves of the first word and the socket number right-justified in the second.

PUPNM performs one of two functions, controlled by B0 of 2 in the call. If B0 is off, PUPNM looks up the address given in the two words pointed to by RH 2, and outputs a string corresponding to that address (unless B4 is on; see below). This conversion is controlled by the other bits in LH 2. If B2 is off and the supplied address does not exactly match some address in the network directory, an error occurs; if B2 is on, however, PUPNM will construct an expression whose value is that address, possibly including octal address constants. For example, the address 3#377#0 will yield an error if B2 is off, but will generate the string "Parc-Net3+377#" if B2 is on.

If B1 is off, the generated string will be an expression with a separate term for each nonzero element in the address. For example, the address 3#200#3 will yield the string "Parc-Net3+Maxc1+FTP". If B2 is on, a term will be omitted if its value is implicitly determined by one or more of the other terms. Hence the address 3#200#3 will yield the string "Maxc1+FTP" since one of the values of "Maxc1" is 3#200#.

If B3 is on and the supplied address exactly matches some address in the network directory, PUPNM returns in RH 3 the byte number in the directory at which that address block starts. This permits a user program to perform any further processing desired, such as obtaining attributes or alternative addresses for the same directory entry. In order to read the network directory, a program must open the file in thawed mode. PUPNM returns in LH 3 the version number of the file <System>Pup-Network.Directory that the index is valid in (which might be different from the highest-numbered version if a new version of the directory is in the process of being created when PUPNM is executed).

If B4 is on and the supplied address exactly matches some address in the network directory, PUPNM looks up the attribute name designated by the string pointer in 4. If the entry has such an attribute, the corresponding attribute value is output to the designator in 1; if no such attribute exists, an error occurs. B4 on suppresses the normal outputting of the name string corresponding to the address and forces B2 to be off.

PUPNM's second function, invoked when B0 is on, is to translate a port name expression (constructed as described in the preceding section) into an address or a list of addresses. For this operation, the source must be a string (i.e. JFNs, terminal designators, etc., are not permitted) terminated by a null, space, rubout, or any control character. '!' also terminates the string when PUPNM is called from monitor mode. PUPNM returns the value of that expression as a list of addresses stored in the table pointed to by 2. B9-17 of 2 in the call specify the length of the table in words; PUPNM returns in LH 2 the number of words actually stored (i.e. two times the number of addresses in the list). If the address list is longer than will fit in the table, LH 2 contains the number of words that would have been stored if the table had been long enough.

If B1 of 2 is on in the call, recognition may be performed on a name string if it is terminated by

null. If such a string is a unique initial substring of some name in the network directory, the remainder of that name is appended to the source string and the string pointer in 1 updated appropriately. An ambiguity causes the error return to be taken with a distinct error code.

The possible errors returned by PUPNM (aside from the usual errors for bad JFNs and the like) are as follows:

PUPNX1	Name or address not found. B0 and B2 were off and the supplied address did not exactly match some address in the network directory, or B0 was on and some name in the source expression was not found.
PUPNX2	Name ambiguous. Recognition was invoked (B0 and B1 on), and the last name in the source string (terminated by null) was an initial substring of more than one name in the network directory.
PUPNX3	Syntax error or illegal address. B0 was off and an address was supplied with network, host, or socket number out of bounds, or B0 was on and the source string was not a legal expression.
PUPNX4	Inconsistent values in name expression. Two terms joined by '+' yielded an empty intersection.
PUPNX5	Syntax error in attribute name string.
PUPNX6	Attribute name not found.

PUPNM JSYS and error code assignments are:

PUPNM	JSYS 443
PUPNX1	602030
PUPNX2	602031
PUPNX3	602032
PUPNX4	602033
PUPNX5	602034
PUPNX6	602035

Network Directory Maintenance

The file <System>Pup-Network.Directory is created by first preparing a text file containing the information to be kept in the directory. This file is <System>Pup-Network.Txt, and it should be consulted for details on the required format.

The text file is compiled into a directory by means of the program MakDir (<System>MakDir.Sav, source in <Pup>MakDir.Mac). MakDir first requests an input filename (default Pup-Network.Txt on the connected directory) and compiles it into internal storage. If errors are detected, appropriate messages are printed and the program quits upon reaching the end of the input file. If no errors are detected, MakDir then requests an output file (default input-filename.Directory on the connected directory) onto which it writes the compiled results.

When compiling the directory that is actually to be installed in the system and distributed to other Name Lookup servers, it is *essential* that MakDir be run while connected to <System>. Creating a Pup-Network.Directory file on another directory and then copying it to <System> will cause the version number contained within the file to be inconsistent with the file's version number in <System>, which will in turn cause the distribution process to malfunction.

When Tenex is started up, it maps in the highest-numbered version of <System>Pup-Network.Directory. To permit installing a new version of the directory without restarting Tenex, a function has been added to the (privileged) OPRFN JSYS. Executing OPRFN with the string

SIXBIT /PUPDIR/ in 1 causes Tenex again to map in the highest-numbered version of the directory. This OPRFN is invoked by the privileged Exec command "Initialize PupDirectory".

Another program, TypDir, interprets a network directory and outputs a formatted dump showing its internal structure. This is handy to have while debugging programs that read the directory itself. TypDir requests an input filename (default Pup-Network.Directory on the connected directory) and an output filename (default input-filename.Lst on the connected directory), and produces a text file whose format should be self-explanatory.