JEW 2-SEP-74 04:29 23904

DRAFT JEW 2 SEP 74 7:35PM

Procedure Call Protocol

(J23904) 2-SEP-74 04:29;;;; Title: Author(s): James E. (Jim) White/JEW; Distribution: /NPG([INFO-ONLY]) JBP([INFO-ONLY]) RWW([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: JEW; Origin: < WHITE, PCP-PCP,NLS;5, >, 2-SEP-74 04:17 JEW ;;;; ####;

DRAFT JEW 2 SEP 74 7:35PM

One of a series of related documents.

INTRODUCTION 1

Several ARPANET applications (i.e, third- or fourth-level) protocols have been designed and implemented since the Host-Host Protocol was adopted in 1970. Most have been bootstrapped from lower-level applications protocols. For example, the File Transfer Protocol (FTP) was built upon TELNET, the Remote Job Entry Protocol upon both TELNET and FTP, and so forth. The highest common denominator of all of these bootstrapped protocols is TELNET. 1a

Although the bootstrapping principle is a sound basis for Network protocol development, the author believes that Telnet is NOT the most appropriate foundation for a large class of applications protocols, offering little more that a character set in all but terminal-driven situations. The author contends that a: 1b

Procedure Call Protocol (PCP) -- a Network-standard mechanism for invoking arbitrary, named, argument-driven and result-producing procedures in a server process... 1b1

is a much more appropriate and powerful foundation for many applications protocols. 1c

The author believes that the adoption by the Network community of a PCP as the basis for most applications protocols would have at least the following effects: 1d

1) expedite the specification of applications protocols by permitting their documentation to have a functional, rather than a syntactic orientation, 1d1

2) largely eliminate the need for separate, application-specific user processes, 1d2

3) reduce the cost of making large, existing software systems available as Network servers by allowing a Network interface more compatible with their internal organization, 1d3

4) provide the basis for a more natural interface between local and remote procedures, and therefore 1d4

5) encourage the sharing of software, by making procedures on remote hosts as accessible to the programmer as local ones. 1d5

The PCP proposed in this document is intended to be suitable both for interlinking procedures on different hosts, and for linking procedures in different processes within a single host. The present document gives only a functional description of PCP, applicable to both classes of use. A subsequent document will provide a detailed, syntactic description of the protocol for its Network application. 1e

The author hereby solicits comments on both PCP and its underlying premise; comments should be addressed to the entire Network Liaison Group. As part of its current National Software Works (NSW) effort, SRI-ARC will implement and employ the PCP described in this document (perhaps modified by suggestions from the Network community) to make the core functions (or "backend") of NLS available both as a Network server process, and as a Tenex fork. SRI-ARC will also implement an interactive NLS "frontend" for PDP-10 TENEX which will use the backend in either of these two modes; and an additional frontend, for the PDP-11 ELF system, which will use the backend via the Network. 1f

The present document is the foundation for a series of subsequent documents describing higher-level tools designed to operate within the setting provided by the Procedure Call Protocol. 1g

THE MODEL 2

Environments 2a

An "environment" is a collection of "procedures" and "data structures" which share a common host machine, instruction set, operating system, and run-time program environment. An environment can run either in parallel with or, by disciplining itself appropriately, in series with another environment. 2a1

PCP provides a mechanism by which two such environments, connected by a logical communication path, can share one anothers procedures. 2a2

Procedures 2b

Introduction 2b1

A procedure is a named body of executable code, residing within a particular environment, which is executed in response to a "call" from another procedure, and which eventually "returns" to its calling procedure. In conventional software systems, in which both calling and called procedures always reside in the same environment, the call-return mechanisms (CRMs) are each usually just a few machine instructions. PCP provides an alternate call-return mechanism to be used when the two procedures reside in different environments. 2b1a

NOTE: Remotely-callable procedures are said to be "external" procedures of the environment in which they reside; all other procedures in the environment, presumably subroutines of external procedures, are called "internal" procedures. PCP places no restrictions upon the CRM employed, within the environment, for dispatching internal procedures, nor for dispatching locally-called external procedures. 2b1a1

Arguments, Modifiers, and Results 2b2

The operation of a procedure is controlled by means of zero or more parameters or "arguments" passed to it by its calling procedure; subsequent operation of the calling procedure is then in turn affected by zero or more "results" produced by the called procedure. The transfer of arguments to, and results from, the called procedure is part of the CRM (and therefore part of PCP). 2b2a

A procedure also accepts zero or more additional parameters, or "modifiers", in addition to whatever arguments it requires, whenever it is called recursively (i.e. by itself). PCP's provision for modifiers is simply a device that permits both retail and wholesale versions of an operation to be implemented and described as a single procedure, rather than as two, slightly different ones. 2b2b

PCP's CRM permits a procedure argument or modifier to be either: 2b2c

1) a data structure (data structures can thus be used as vehicles of communication, as well as storage) provided by the calling procedure, 2b2c1

2) an external data structure in the called procedure's environment, or 2b2c2

3) an "attribute" of an external data structure in the called procedure's environment. 2b2c3

Result Dispositions 2b2d

A procedure's results can be used in a variety of ways by the caller: they can be examined or manipulated, stored for later use, used immediately as arguments to other procedures, or simply ignored. 2b2d1

In conventional software systems, the cost of returning a procedure result to the calling procedure is low, involving at worst a main storage transfer. The CRM can therefore afford to blindly return the results of a procedure to the caller, and leave their disposition to him. 2b2d2

But when the procedure and its caller reside in different environments, the cost of returning results is significantly higher. And if after having been retrieved at such cost, the result is then ignored by the calling procedure, or worse still, returned to the same environment as an argument to a subsequent procedure, the increased cost is (at least emotionally) very hard to bear. 2b2d3

To help eliminate such inefficiencies, PCP's CRM can be pre-instructed as to the procedure results' intended use, or "disposition". The CRM permits a procedure result (which, like an argument or modifier, is a data structure) to be either: 2b2d4

1) returned to the calling procedure (if its disposition is RETURN), 2b2d4a

2) stored automatically in an external data structure in the called procedure's environment (if the disposition is a reference to a data structure), 2b2d4b

3) used automatically to modify an attribute of an external data structure in the called procedure's environment (if the disposition is a reference to the attribute of a data structure), or 2b2d4c

4) discarded immediately by the called procedure's environment (if its disposition is DISCARD). 2b2d4d

In the last three cases, the CRM returns an EMPTY result to the caller. 2b2d5

Outcomes 2b3

Many procedures attempt well-defined tasks at which they either succeed or fail. The result of such a procedure can be communicated to the caller in a variety of ways: it can be returned as a formal result of the procedure, stored in a named data structure, or registered in any other way agreed upon by both caller and callee. Although it cannot prevent the use of such ad hoc schemes for reporting the "outcome" of a remote procedure, PCP provides and encourages the use of a more standard mechanism, by allowing for the return of a special meta-result along with the normal procedure results. 2b3a

The outcome meta-result may have any of the following values, each with the indicated implication: 2b3b

SUCCESS: the procedure has succeeded at its task. 2b3b1

FAILURE: the procedure has failed at its task. 2b3b2

NEUTRAL: no indication of success or failure is returned. 2b3b3

ERROR: the procedure has encountered an unexpected and irrecoverable error, and has returned a program-readable error code, and an optional human-readable diagnostic message, in place of the normal procedure results. 2b3b4

Transfer and Return of Control 2b4

A procedure normally receives control from its caller, performs its task, and then relinquishes control. More complicated control transfer sequences are sometimes required, and are therefore supported by PCP. Besides providing for the normal call-return sequence just described, PCP permits a remote procedure to summon assistance from any of the procedures along its control thread, to notify any such procedure of an arbitrary event, or to interact with its caller as a co-routine. 2b4a

When a remote procedure returns control to its caller, it specifies the condition or "terms" under which it does so. The terms of the return are transmitted to the calling procedure as a second meta-result, which may have any of the following values, each with the indicated implication: 2b4b

EXIT: the procedure has completed execution, and is returning final control to its caller, along with the outcome meta-result and the procedure's results. This is the normal return, described in the preceeding section. 2b4b1

RPRT: the procedure has not yet completed execution, but is returning temporary control to its caller to report intermediate status and results, and expects to be resumed with an indication of whether or not to proceed, and with optional new arguments. This is the co-routine return. 2b4b2

NOTE: the procedure has not yet completed execution, but is returning temporary control to its caller to notify it of a particular event, and expects to be resumed after the calling procedure has acted upon it. In general, the caller is obligated to propagate the notice up the thread of control (by returning to ITS caller under the same terms) before resuming the remote procedure. 2b4b3

HELP: the procedure has not yet completed execution, and cannot proceed without first obtaining assistance from its caller with a particlar problem. The remote procedure expects to be resumed with an indication of whether or not the requested help has been provided. In general, if the caller is

incapable of helping the remote procedure, it is obligated to propagate the request up the thread of control (by returning to ITS caller under the same terms) and then to return to the remote procedure whatever help the higher-level procedure has provided. 2b4b4

WKNG: these terms are reserved for use by the Environmental Control Package. 2b4b5

PCP requires that a calling procedure eventually resume a procedure which returns to it on any terms other than EXIT. 2b4c

Syntax Conventions 2b5

Procedure descriptions in all PCP-related documents will have the following format: 2b5a

> Terse statement of procedure's function 2b5a1
> > Name-of-procedure (arguments [modifiers] -> results) 2b5a1a
> > Verbose description of the procedure's function, the arguments and modifiers it requires, and the results it returns. 2b5a1b
> > Argument/result types: 2b5a1c
> > > A detailed description of arguments, modifiers, and results, including their types and, where appropriate, acceptable values. 2b5a1c1

Throughout the procedure descriptions, the terms "host" and "invoking" will refer, respectively, to the environment that contains the procedure, and the environment from which the procedure is called. 2b5a2

Data Structures 2c

Introduction 2c1

Apart from its use as a communication vehicle, a data structure is a named data store, resident in a particular environment, which holds environment state information, and which exists throughout the environment's lifetime. In conventional software systems, in which both the data structure and the procedures that manipulate it always reside in the same environment, the read-write mechanism (RWM) is usually just a few machine instructions. PCP provides an alternate RWM to be used when the data structure and the procedure desiring to manipulate it reside in different environments. 2c1a

NOTE: Remotely-manipulable data structures are said to be "external" data structures of the environment in which they reside, and all other data structures in the environment are called "internal" data structures. PCP places no restrictions upon the RWM employed, within the environment, for manipulating internal data structures, nor for manipulating external data structures locally. 2c1a1

Data Types 2c2

Introduction 2c2a

When transmitted between environments (as a procedure argument, modifer, or result), the value of a data structure must, in general, be represented in some standard format convenient for and agreed upon by both sending and receiving environments (whose machines may, for example, have different word lengths and character sets). The sender's CRM (i.e. PCP) must, therefore, encode the data structure in the standard format, and the receiver's CRM (PCP again) must decode it. 2c2a1

[It should be clear that, because of the required encoding and decoding, remote procedure arguments, modifiers, and results are effectively passed by value, rather than by name. Hence, procedure "results" cannot be returned by way of modified arguments.] 2c2a1a

To encode or decode a data structure for transmission, the CRM must know: 2c2a2

1) its "type" (e.g. integer, string, or list) -- somehow communicated to it by the sending procedure, 2c2a2a

2) its internal representation -- a characteristic of the sending environment, and 2c2a2b

3) the standard format for data structures of its type -- specified by PCP. 2c2a2c

To facilitate the task of encoding and decoding data structures for transmission, a limited set of data types are defined by PCP. Every argument, modifier, or result of a remote procedure must be of one of the types permitted by PCP. 2c2a3

Types 2c2b

The following types of data structures are currently supported by PCP: 2c2b1

- STRING: a text string, with both a current and a maximum length (in characters), 2c2b1a
- INTEGER: a signed integer, 2c2b1b
- WORD: an unsigned integer capable of being stored in an address space word (used by the Low-Level Debug Package, described in another document), 2c2b1c
- BOOLEAN: true or false, 2c2b1d
- EMPTY: null, 2c2b1e
- LIST: an ordered set of data structures, with both a current and a maximum number of elements. This last data type provides a mechanism by which arbitrarily complex "composite" data structures can be constructed from the other, "atomic" types listed above, 2c2b1f

In all PCP-related documents, the following syntax will be used to describe a data structure (brackets surround optional elements): 2c2b2

```
datastrucdesc ::= [name ':] [use '%] typelist          2c2b2a
typelist      ::= type ['[ valuelist ']] ['/ typelist] 2c2b2b
type          ::= "STRING" / "INTEGER" / "WORD" /
                  "BOOLEAN" / "EMPTY" / "LIST" ['(
                  list ')] / "any"                     2c2b2c
list          ::= datastrucdesc [', (list/"...")]      2c2b2d
valuelist     ::= value ['/ valuelist]                 2c2b2e
value         ::= [symb] ['= abs]                      2c2b2f
```

where: 2c2b3

- NAME is the structure's name 2c2b3a
- USE denotes the structure's function 2c2b3b
- ANY implies that the structure may be of any type 2c2b3c
- SYMB is a symbolic name for the data structure's value 2c2b3d
- ABS is the data structure's value in absolute form 2c2b3e
- ... implies zero or more list elements like the previous one 2c2b3f
- / separates alternatives 2c2b3g

Shorthands for Some Recurring Complex Data Types 2c2c

Strictly as a convenience in describing certain data structures, we define the following shorthands or pseudo data structure types (whose names are suffixed with an asterisk ('*)): 2c2c1

1) To denote either a single element or a list of elements, assured that the element is not itself a list: 2c2c1a

LIST* (element) ==> element / LIST (element, ...) 2c2c1a1

2) To denote a reference to an attribute of an external data structure, qualified by its package name: 2c2c1b

DATAREF* ==> pkname: strucattrib% STRING 2c2c1b1

Attributes 2c3

Introduction 2c3a

An attribute of a data structure A is itself a data structure whose value is some simple characteristic of A. An attribute is thus effectively a PCP builtin function which maps one data structure into another. Attributes may be applied recursively. 2c3a1

Types 2c3b

The following data structure attributes are currently supported by PCP: 2c3b1

VALUE: denotes a data structure whose type and value are those of another data structure. This attribute type effectively permits implicit argument transmission. 2c3b1a

DIMEN: denotes a data structure of type INTEGER whose value is the current number of elements in another data structure of type LIST, or (when applied to a data stucture of any other type) one. 2c3b1b

MAXDIM: denotes a data structure of type INTEGER whose value is the maximum number of elements in another data structure of type LIST, or (when applied to a data stucture of any other type) one. 2c3b1c

LENGTH: denotes a data structure of type INTEGER whose value is the current length in characters of another data structure of type STRING. 2c3b1d

MAXLEN: denotes a data structure of type INTEGER whose value is the maximum possible length in characters of another data structure of type STRING. 2c3b1e

ELEM: denotes a data structure whose type and value are those of a specified element of another data structure of type LIST. If the attribute is applied to a data structure of any other

type, the latter is treated as a single-element list. 2c3b1f

In all PCP-related documents, the following syntax will be used to describe a reference to a data structure attribute: 2c3b2

```
strucattrib ::= ("VALUE" / "DIMEN" / "MAXDIM" /
                 "LENGTH" / "MAXLEN" / "ELEM" elemid)
                '( strucname / datastrucdesc /
                strucattrib ')                          2c3b2a
elemid      ::= elemindx / elemname                     2c3b2b
```

where: 2c3b3

STRUCNAME is the name of the data structure. 2c3b3a

ELEMINDX is the element's ordinal position (beginning with one) within the list. 2c3b3b

ELEMNAME is the element's name. If ELEMNAME is ambiguous, the element with lowest ELEMINDX is selected. 2c3b3c

Prototypes 2c4

It is sometimes desireable to transmit, as an argument, modifier, or result, the form, but not the content, of a data structure. For example, a remote procedure which allocates space for a temporary data structure might require, as one of its arguments, an example of the kind of data structure to be subsequently stored in that space. 2c4a

A "prototype" is simply a data structure whose "shape and size" are noted by the receiving procedure, but whose value (or values, in the case of a LIST) is ignored. The value of a prototype should be chosen by the sender to minimize the expense of its transmission (e.g. by setting the length -- the current, not the maximum length -- of a string to zero). 2c4b

Syntax Conventions 2c5

Data structure descriptions in PCP-related documents will have the following format: 2c5a

> Name-of-data-structure Terse statement of data structure's use 2c5a1

> Verbose description of the data structure's use. 2c5a2

> Data structure type: 2c5a3
> A detailed description of the data structure. 2c5a3a

THE IMPLEMENTATION 3

Inter-Environment Communication 3a

Introduction 3a1

PCP assumes that a logical data path can be created between two environments, and that each environment provides to its local procedures the primitives necessary to manipulate it. That is, the existence of some form of inter-environment communication (IEC) is assumed, not provided, by PCP. 3a1a

since PCP is designed to be useful for mediating communication between different kinds of environments, the peculiarities of the environment interface are assumed hidden in and accounted for by IEC. 3a1b

What follows is a description of the IEC primitives assumed (i.e. required) of an environment which seeks to communicate with remote environments via PCP, and of their use by PCP. 3a1c

Primitives 3a2

Open path to environment 3a2a

OPNPTH (envname => pathndle) 3a2a1

This primitive creates a full duplex logical data path between the current environment and the environment identified by ENVNAME, and returns to the caller a handle PATHNDLE to the data path, for use in subsequent IEC primitives. 3a2a2

This primitive is used by the Environmental Control Package, described in another document, to create an inferior environment. 3a2a3

Argument/result types: 3a2a4

envname - STRING 3a2a4a
pathndle- any 3a2a4b

Close path to environment 3a2b

CLSPTH (pathndle) 3a2b1

This primitive deletes the data path identified by PATHNDLE, previously created via OPNPTH. 3a2b2

This primitive is used by the Environmental Control Package, described in another document, to delete an inferior environment. 3a2b3

Argument/result types: 3a2b4

pathndle- any 3a2b4a

Send data structure to environment 3a2c

SNDPTH (pathndle, datastrucencde) 3a2c1

This primitive transmits an encoded data structure DATASTRUCENCDE to the remote environment along the data path identified by PATHNDLE. 3a2c2

PCP uses this, and the RCVPTH primitive described below, primitive to transmit a procedure call or return request to a remote environment. 3a2c3

Argument/result types: 3a2c4

pathndle - any 3a2c4a
datastrucencde- any 3a2c4b

Accept data structure from environment 3a2d

RCVPTH (pathndle => datastrucencde) 3a2d1

This primitives accepts delivery of the next encoded data structure DATASTRUCENCDE transmitted by the remote environment along the data path identified by PATHNDLE. 3a2d2

Argument/result types: 3a2d3

pathndle - any 3a2d3a
datastrucencde- any 3a2d3b

Test for data structure from environment 3a2e

TSTPTH (pathndle) 3a2e1

This primitive determines whether or not there exists a data structure, sent by the remote environment on the path identified by PATHNDLE, awaiting immediate delivery. If a subsequent RCVPTH primitive can be immediately satisfied, the primitive returns an outcome of SUCCESS; otherwise, it returns FAILURE. 3a2e2

The Environmental Control Package uses this primitive to test for the completion of a parallel procedure. 3a2e3

Argument/result types: 3a2e4

pathndle- any 3a2e4a

Signal environment 3a2f

SIGPTH (pathndle, bit) 3a2f1

This primitive transmits a single bit BIT of information to the remote environment attached to the path PATHNDLE, awakening or interrupting it as necessary. 3a2f2

The Environmental Control Package, described in another document, uses this primitive to implement its $INTEVM and $RSMEVM procedures. 3a2f3

Argument/result types: 3a2f4

pathndle- any 3a2f4a
bit - BOOLEAN 3a2f4b

Implementation 3a3

To completely specify the interconnection of any specific pair of environments via PCP, one must more precisely specify the IEC primitives described above. In particular, one must specify the format of DATASTRUCENCDE, and the manner in which the IEC primitives are to be constructed from still more primitive operations. This much smaller task will be undertaken in subsequent documents for at least the following classes of environment pairs: 3a3a

1) an ARPANET server process and any of its user processes, 3a3a1

2) a Tenex fork and any of its inferior forks. 3a3a2

PCP Communiques 3b

Introduction 3b1

The PCP CRMs of connected environments communicate with one another via the data structures, or "communiques", they exchange by means of the IEC SNDPTH and RCVPTH primitives described above. The three defined communiques are described below. 3b1a

Though they appear in communiques, "package identifiers", or PKIDs, are not used by PCP proper. A package is a construct of the PCP Support package, described in another document. When PCP is used outside of the higher-level framework provided by the Support package, PKIDs should be set to zero wherever they are required. 3b1b

Communiques 3b2

Call procedure 3b2a

CALL (pkid, pname, args, mods, disp) 3b2a1

This communique requests that the receiving environment call the procedure PNAME in package PKID, with arguments ARGS and modifiers MODS, on behalf of the sending environment. 3b2a2

If the remote procedure returns on terms EXIT (with outcome other than ERROR) or RPRT, DISP specifies the disposition of the procedure's results -- either a single disposition for all results, or (if DISP is a LIST) a separate disposition for each result. 3b2a3

If the remote procedure makes a return on terms NOTE, HELP, RPRT, or WKNG, it is obliged to provide the CID required for the procedure's subsequent reentry. 3b2a4

Format: 3b2a5

LIST (op, pkid, pname, args, mods, disp) 3b2a5a

op = INTEGER [CALL=0] 3b2a5b
pkid = INTEGER 3b2a5c
pname= STRING 3b2a5d
args = LIST 3b2a5e

```
        mods - LIST                                                    3b2a5f
        disp - LIST* (INTEGER [RETURN=0 / DISCARD=1] /
               dataref% DATAREF*)                                      3b2a5g

Resume procedure                                                       3b2b

   RESUME (cid, parms, disp)                                           3b2b1

   This communique requests that the receiving environment
   resume the procedure identified by CID (which must have
   previously returned on terms NOTE, HELP, RPRT, or WKNG),
   with parameters PARMS,                                              3b2b2

   If the remote procedure returns on terms EXIT (with
   outcome other than ERROR) or RPRT, DISP specifies the
   disposition of the procedure's results (as for CALL).               3b2b3

   Format:                                                             3b2b4

      LIST (op, cid, parms, disp)                                      3b2b4a

      op    - INTEGER [RSME=1]                                         3b2b4b
      cid   - INTEGER                                                  3b2b4c
      parms- LIST                                                      3b2b4d
      disp - LIST* (INTEGER [RETURN=0 / DISCARD=1] /
             dataref% DATAREF*)                                        3b2b4e
```

Return from procedure 3b2c

RETURN (terms, outcome, results, cid) 3b2c1

This communique acknowledges the return of a previously initiated (or resumed) procedure on the indicated TERMS. 3b2c2

Unless TERMS is EXIT, the CID provided becomes the receiving environment's basis for resuming the remote procedure. 3b2c3

Format: 3b2c4

```
LIST (op, terms, outcome, results, cid)                    3b2c4a

op    = INTEGER [RTN=2]                                    3b2c4b
terms= INTEGER [EXIT=0 / RPRT=1 / NOTE=2 / HELP=3 /
       WKNG=4]                                             3b2c4c
   EXIT: outcome= INTEGER [SUCCESS=0 / FAILURE=1 /
                  NEUTRAL=2 / ERROR=3]                     3b2c4c1
         ERROR: results= LIST (errcode% INTEGER,
                         errmsg% STRING)                   3b2c4c2
         other: results= any                               3b2c4c3
   RPRT: outcome= status% INTEGER                          3b2c4c4
         results= intres% any                              3b2c4c5
   NOTE: outcome= event% INTEGER                           3b2c4c6
         results= eventdesc% any                           3b2c4c7
   HELP: outcome= condition% INTEGER                       3b2c4c8
         results= conddesc% any                            3b2c4c9
   WKNG: outcome= unused% EMPTY                            3b2c4c10
         results= unused% EMPTY                            3b2c4c11
cid   = INTEGER / EMPTY                                    3b2c4d
```

DRAFT PCP
The Procedure Call Protocol

3-SEP-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

PCP is an inter-environment procedure call and return mechanism which provides a setting in which higher-level tools can be remotely offered and used.

(J23905) 2-SEP-74 04:32;;;; Title: Author(s): James E. (Jim) White/JEW; Distribution: /NPG([INFO-ONLY]) JBP([INFO-ONLY]) RWW([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: JEW;
Origin: < WHITE, PCP-PCPSUP,NLS;8, >, 2-SEP-74 02:33 JEW ;;;; ####;

One of a series of related documents.

INTRODUCTION 1

The PCP Support Package (package name=$PCPSUP) contains those procedures and data structures which a remote environment requires to use the host environment conveniently. $PCPSUP includes procedures for opening and closing packages, manipulating data structures within the host environment, creating temporary ones, for logging into and resetting the host environment, and a NOP. It contains data structures with menus of the host environment's supported packages, and their external procedures and data structures. 1a

Packages 1b

The external procedures and data structures within an environment are partitioned, by function, into one or more "packages". Packages are referred to initially (in the $OPNPK procedure) by name, and thereafter via a "package identifier", or PKID. The entire contents of a package are accessible to another environment if and only if it has successfully "opened" the package (i.e. if it has obtained a PKID for it). 1b1

NOTE: The PCP Support Package itself is always considered open (with PKID=0) and need not, indeed cannot, be explicitly opened or closed (with $OPNPK and $CLSPK). 1b1a

A Package Programmer's Guide (PPG), like the current document, is assumed to exist for each package implemented, and to contain: 1b2

1) a description of the package and its function, 1b2a
2) a description of each external procedure, including its name, function, the type and function of each of its arguments and modifiers, the type and function of each of its results, and any accessibility peculiarities, 1b2b
3) the name, type, function, and accessibility peculiarities of each external data structure, 1b2c

PROCEDURES 2

Open package 2a

sOPNPK (pkname => pkid) 2a1

This procedure opens the host environment's package PKNAME, and makes it known (and its external procedures and data structures accessible) to the invoking environment via the handle PKID. 2a2

Argument/result types: 2a3

pkname- STRING 2a3a
pkid - INTEGER 2a3b

Close package 2b

sCLSPK (pkid) 2b1

This procedure closes the host environment's previously-opened package, known to the invoking environment via PKID, and makes its procedures and data structures inaccessible to the invoking environment. 2b2

Argument/result types: 2b3

pkid- INTEGER 2b3a

Read data structure attribute 2c

$RDDATA (dataref -> value) 2c1

This procedure returns the value VALUE of an attribute DATAREF of an external data structure in one of the host environment's previously-opened packages (implicitly named by DATAREF). 2c2

Argument/result types: 2c3

dataref- DATAREF* 2c3a
value - any 2c3b

Write data structure attribute 2d

$WRDATA (dataref, value) 2d1

This procedure assigns a new value VALUE to an attribute DATAREF of an external data structure in one of the host environment's previously-opened packages (implicitly named by DATAREF). 2d2

BY definition, the MAXLEN or MAXDIM attribute of a data structure is immune to $WRDATA. 2d3

Argument/result types: 2d4

dataref- DATAREF* 2d4a
value - any 2d4b

Create temporary data structure 2e

$CRTTMP (tmpname, tmppro) 2e1

This procedure creates a temporary external data structure TMPNAME, like the prototype TMPPRO, in the host environment's PCP Support Package. Once created, the temporary data structure can be used and manipulated like any other external data structure. 2e2

Argument/result types: 2e3

tmpname- STRING 2e3a
tmppro - any 2e3b

Delete temporary data structure 2f

$DELTMP (tmpname) 2f1

This procedure deletes the temporary data structure TMPNAME, previously created via $CRTTMP, from the host environment's PCP Support Package. 2f2

Argument/result types: 2f3

tmpname- STRING 2f3a

Reset environment 2g

$RESET () 2g1

This procedure resets the host environment to its creation state. If the environment supports the Environment Control Package, described in another document, it deletes all inferior environments and releases all VIDs obtained via $GETHND. 2g2

Login environment 2h

$LOGIN (user, password, acct) 2h1

This procedure associates the host environment's use with a user USER (for access-control purposes), protected by the password PASSWORD, and an account ACCT (for billing purposes). 2h2

Argument/result types: 2h3

user = STRING 2h3a
password= STRING 2h3b
acct = STRING / EMPTY 2h3c

No operation 2i

$NOP (argument => argument) 2i1

This procedure is a NOp, simply echoing its argument as its result. It can be called remotely to verify the communication path to, and proper functioning of the host environment. 2i2

Argument/result types: 2i3

argument= any 2i3a

DATA STRUCTURES 3

$PKAGES List of offered packages 3a

This read-only, reader-dependent data structure is a list of the packages, within the host environment, available to the invoking environment, and specifies for each: its name PKNAME, and, if opened by the invoking environment, its PKID. 3a1

Data structure type: 3a2

$pkages: LIST (pkname: pkid% INTEGER / EMPTY, ...) 3a2a

$EXTPRC Lists of external procedures 3b

This read-only, reader-dependent data structure is a list of the external procedures contained in each of the host environment's open packages PKNAME, and contains for each procedure: its name PRCNAM, a list ARGPTS of prototype arguments, a list MODPTS of prototype modifiers, and a list RESPTS of prototype results. 3b1

Data structure type: 3b2

$accprc: LIST (pkname: LIST (prcnam: LIST (argpts% LIST, modpts% LIST respts% LIST), ...), ...) 3b2a

$EXTDAT Lists of external data structures 3c

This read-only, reader-dependent data structure is a list of the external data structures contained in each of the host environment's open packages PKNAME, and contains for each data structure: its name DATNAM and a prototype DATPT. 3c1

Data structure type: 3c2

$accdat: LIST (pkname: LIST (datnam: datpt% any, ...), ...) 3c2a

APPENDIX: PSEUDO IMPLEMENTATIONS 4

Introduction 4a

The following are implementations of some of SPCPSUP's procedures in something resembling SRI-ARC's L10 programming language. Their purpose is to help clarify the procedure definitions and to suggest, in broad terms, an implementation strategy. 4a1

Selected Procedures 4b

```
($opnpk) %open package%
PROCEDURE (pkname : pkid);                                          4b1
   %declarations%                                                   4b1a
      LOCAL INTEGER i;                                              4b1a1
      LOCAL POINTER pack, port;                                     4b1a2
   %verify access to package%                                       4b1b
      pack _ findpack (pkname=pkname);                              4b1b1
      port _ findport (status=active & vid=masters [LENGTH
      (masters)]);                                                  4b1b2
      LOOP                                                          4b1b3
         BEGIN                                                      4b1b3a
         FOR i_1 TO LENGTH (port.pklist) DO                         4b1b3b
            IF port.pklist [i] = pack.pkid THEN EXIT LOOP 2;        4b1b3b1
         $ABORT (code, "No access to package.");                    4b1b3c
         END;                                                       4b1b3d
   %check for package already open%                                 4b1c
      FOR i_1 TO LENGTH (port.opnpklist) DO                         4b1c1
         IF port.opnpklist [i] = pack.pkid THEN                     4b1c1a
            $ABORT (code, "Package already open.");                 4b1c1a1
   %add PKID to port's list%                                        4b1d
      port.opnpklist [BUMP LENGTH (port.opnpklist)] _
      pack.pkid;                                                    4b1d1
      BUMP pack.opncnt;                                             4b1d2
   %return%                                                         4b1e
      $EXIT (success, pack.pkid);                                   4b1e1
      END.                                                          4b1e2
```

```
(sclspk) %close package%
PROCEDURE (pkid);                                                    4b2
   %declarations%                                                    4b2a
      LOCAL INTEGER i;                                               4b2a1
      LOCAL POINTER pack, port;                                      4b2a2
   %assure package open%                                             4b2b
      pack _ findpack (pkname=pkname);                               4b2b1
      port _ findport (status=active & vid=masters [LENGTH
      (masters)]);                                                   4b2b2
      LOOP                                                           4b2b3
         BEGIN                                                       4b2b3a
         FOR i_1 TO LENGTH (port.opnpklist) DO                       4b2b3b
            IF port.opnpklist [i] = pkid THEN EXIT LOOP 2;           4b2b3b1
         $ABORT (code, "Package not open.");                         4b2b3c
         END;                                                        4b2b3d
   %delete PKID from port's list%                                    4b2c
      port.opnpklist [i] _ port.opnpklist [LENGTH
      (port.opnpklist)];                                             4b2c1
      BUMP DOWN LENGTH (port.opnpklist);                             4b2c2
      BUMP DOWN pack.opncnt;                                         4b2c3
   %return%                                                          4b2d
      $EXIT (success);                                               4b2d1
      END.                                                           4b2d2
```

```
Internal Global Data Structures                                          4c

   GLOBAL RECORD port                                                    4c1

      INTEGER vid,       %VID to which this port corresponds%            4c1a
      INTEGER status,    %status of this VID%                            4c1b

         %free   -- no environment associated with VID%                  4c1b1
         %active -- environment assigned to VID%                         4c1b2
         %dead   -- environment assigned to VID but it's dead%           4c1b3

      INTEGER type,      %relationship of remote environment%            4c1c

         %self     -- the host environment%                              4c1c1
         %superior -- host environment's direct superior%                4c1c2
         %inferior -- a direct inferior%                                 4c1c3
         %head     -- a VID obtained (as headvid) via $GETHND%           4c1c4
         %link     -- support for $GETHND VIDs known elsewhere%          4c1c5
         %tail     -- a VID obtained (as tailvid) via $GETHND%           4c1c6

      INTEGER bckloc,    %VID of previous environment in chain%          4c1d
         %self -- if this is the head%                                   4c1d1
      INTEGER bckrem,    %VID of link element in previous
      environment%                                                       4c1e
         %self -- if this is the head%                                   4c1e1
      INTEGER forloc,    %VID of next environment in chain%              4c1f
         %self -- if this is the tail%                                   4c1f1
      INTEGER forrem,    %VID of link element in next environment%       4c1g
         %self -- if this is the tail%                                   4c1g1
      STRING  envname,   %environment name%                              4c1h
         %null -- unless direct inferior%                                4c1h1
      INTEGER compath,   %IEC path handle%                               4c1i
         %null -- (for now) unless direct inferior/superior%             4c1i1
      INTEGER entcnt,    %# sent CALLs still awaiting EXITs%             4c1j
      BOOLEAN wkng,      %TRUE if sent CALL still awaiting RETURN%       4c1k
      LIST    pklist,    %PKIDs for accessible local packages%           4c1l
      LIST    opnpklist; %PKIDs for open local packages%                 4c1m

   GLOBAL RECORD pack                                                    4c2

      INTEGER pkid,    %PKID to which this package record
      corresponds%                                                       4c2a
      STRING  pkname,  %package name%                                    4c2b
      INTEGER opncnt;  %number of environments with package open%        4c2c

   GLOBAL LIST masters; %VIDs of calling environments%                   4c3
      %top element -- VID of controlling environment%                    4c3a
```

DRAFT $PCPSUP
The PCP Support Package

3-SEP-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

$PCPSUP is a procedure call support tool that operates within the setting provided by the Procedure Call Protocol (PCP -- xxxxxx,), with which the reader of the present document is assumed familiar.

DRAFT JEW 2 SEP 74 7:39PM The Environmental Control Package

(J23906) 2-SEP-74 04:34;;;; Title: Author(s): James E. (Jim)

White/JEW; Distribution: /NPG([INFO-ONLY]) JBP([INFO-ONLY]) RWW([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: JEW;
Origin: < WHITE, PCP-EVMCTL,NLS;10, >, 2-SEP-74 01:16 JEW ;;;;
####;

DRAFT JEW 2 SEP 74 7:39PM The Environmental Control Package

One of a series of related documents.

INTRODUCTION 1

PCP makes possible the orderly construction of large, Network-based, distributed software systems, each component implemented on a different machine, with a different operating system, in a different programming language. 1a

The Environmental Control Package (package name = $EVMCTL) provides the necessary tools for interconnecting two or more environments to form a coherent, Multi-Environment Software System (MESS). $EVMCTL contains those procedures and data structures required to construct from a single "root" environment, manipulate, and dismantle a multi-environment system. The package includes procedures for creating, deleting, logically interconnecting, and relaying procedure calls between environments, and for interrupting and resuming environments, along with the necessary support data structures. 1b

The Multi-Environment Software System 1c

The Environment Tree 1c1

Let MESS be, at any point in time, a tree structure of environments. Every MESS begins with a single, pre-existent root environment to which all other environments are ultimately subordinate. During the course of its execution, the root environment creates one or more subordinate environments, one or more of which may create subordinate environments of their own, and so forth. 1c1a

An environment is said to be the "direct inferior" of the environment that created it, and the "indirect inferior" of each environment further up in the environment tree. 1c1a1

An environment is said to be the "direct superior" of each environment it creates, and the "indirect superior" of each environment further down in the tree. 1c1a2

An environment may have, at any point in time, an arbitrary number of directly inferior environments, but (of course) only one directly superior environment. 1c1a3

Environment Names 1c2

An existing environment creates a direct inferior by offering its "environment name" to the $CRTEVM procedure. Environment names which begin with a dollar sign ('$) are reserved for MESS-wide assignment; all others are available for local assignment. The following MESS-standard generic environment names are currently defined: 1c2a

1) "$F" <SP> filename 1c2a1

The environment created from the executable file FILENAME in the host environment's file system, and run on the host environment's machine. 1c2a1a

2) "$N" <SP> host <SP> socket 1c2a2

The environment run on the ARPANET host HOST (a standard host name or decimal host address), and created by the remote system in response to an ICP to contact socket SOCKET (specified in decimal). 1c2a2a

Known Environments 1c3

Once an environment has been created, it is referenced from another environment by means of an "environment identifier", or VID. A VID is a local handle to an environment; it is always evaluated relative to the environment in which it is used. An environment B is said to be "known" to another environment A if and only if A has a handle to B. $EVMCTL assures that if B is known to A, then A is known to B as well. 1c3a

An environment's direct superior is always known to it (via a special VID whose value is SUPER=-1). A VID is assigned to each direct inferior at its creation. VIDs for other environments must be explicitly obtained via $EVMCTL's $GETHND procedure. An environment is always known to itself via the special VID whose value is SELF=0. 1c3a1

An environment may call remote procedures in any environment known to it. 1c3b

Configuring the MESS 1c4

Viewed one way, a MESS is simply a collection of procedures and data structures partitioned among some arbitrary number of environments. Since any procedure can call both local and remote procedures and manipulate both local and remote data structures, the system's procedures and data structures could, in principle, be partitioned among environments in arbitrary fashion. 1c4a

In practice, however, the programmer must assume that calling a remote procedure or manipulating a remote data structure is more expensive, in terms of both the real and processing time required, than calling or manipulating a local one -- an operation which may be as simple as a single machine instruction. The process of partitioning the system's components among environments must therefore be done with intelligence and care. 1c4b

Serial and Parallel Operation 1c5

BY means of the $CALL and $RESUME procedures, $EVMCTL provides an invoking environment A with a choice between serial or parallel execution of a procedure in an environment C that is remote with respect to the host environment B. 1c5a

When a procedure in C is called serially, its execution is completed (i.e. it signals a return to B) before the $CALL (or $RESUME) procedure in B signals a return to A. 1c5a1

When the procedure in C is called in parallel, environment B signals a return to A on terms WKNG, immediately after initiating the procedure call request at C. Environment A is then free to call additional procedures in B, provided that their execution doesn't require a call upon a procedure in now occupied environment C. 1c5a2

As B returns (on terms WKNG) to A, it returns a CID by which the uncompleted procedure call can be referenced. 1c5a2a

At any time, environment A is free to test for the completion of the procedure running in C by calling the $RESUME primitive in B in parallel mode.

specifying the CID. $RESUME will return, like the original $CALL, on terms WKNG if the procedure is still incomplete, and on some other terms otherwise. 1c5a2b

Alternately, environment A may commit itself to the parallel procedure's completion (as if it had originally initiated it serially) by $RESUMEing it serially, rather than in parallel. 1c5a2c

PROCEDURES 2

Create environment 2a

$CRTEVM (envname => vid) 2a1

This procedure attaches an instance of the environment ENVNAME as a direct inferior of the host environment, and makes it known to the host environment via the handle VID. 2a2

Argument/result types: 2a3

envname= STRING 2a3a
vid = INTEGER 2a3b

Delete environment 2b

$DELEVM (vid) 2b1

This procedure detaches from the host environment and discards, the environment known to it via VID and previously attached as a direct inferior of the host environment via $CRTEVM. 2b2

Argument/result types: 2b3

vid= INTEGER 2b3a

Obtain environment handle 2c

sGETHND (vidlist [backlink] => headvid, tailvid) 2c1

This procedure returns a handle HEADVID for, and thereby makes known to the host environment (provided BACKLINK is absent), a target environment which is presently known indirectly (and therefore not really at all) via the chain of VIDs VIDLIST. In addition, the host environment is made known to the target environment via TAILVID. Whenever possible, a direct communication path is constructed between the two environments. 2c2

The last VID in VIDLIST is evaluated relative to the host environment; every other VID, VIDLIST [i], is evaluated relative to the environment identified by VIDLIST [i+1]. 2c3

The presence of the modifier implies that HEADVID is to be defined, not for use by procedures within the host environment, but for relaying procedure calls to adjacent environments: to the environment known to the invoking environment via the VID BACKLINK (in the backward direction), and to the environment known via VIDLIST [LENGTH (VIDLIST) - 1] to the environment known to the host environment via VIDLIST [LENGTH (VIDLIST)] (in the forward direction). 2c4

Argument/result types: 2c5

vidlist = LIST (INTEGER, ...) 2c5a
backlink= INTEGER 2c5b
headvid = INTEGER 2c5c
tailvid = INTEGER 2c5d

Release environment handle 2d

sRELHND (vid [backlink]) 2d1

This procedure releases the environment handle VID previously obtained with $GETHND. Once the procedure is executed, the environment previously known or chained to the host environment via VID is no longer known or chained to it. 2d2

The modifier must be present if and only if it was present in $GETHND when the VID was obtained. 2d3

Argument/result types: 2d4

vid - INTEGER 2d4a
backlink- INTEGER 2d4b

InterruPt environment 2e

sINTEVM (vid) 2e1

This procedure interrupts the environment known to the host environment via VID, and causes it to save its state on a stack, for subsequent retrieval via $RSMEVM. 2e2

Argument/result types: 2e3

vid- INTEGER 2e3a

Resume environment 2f

sRSMEVM (vid) 2f1

This procedure resumes the previously interrupted environment known to the host environment via VID, causing it to first restore its state from the stack. 2f2

Argument/result types: 2f3

vid- INTEGER 2f3a

Call remote procedure 2g

$CALL (vid, pkid, pname, args, mods, disp, mode [dummy] =>
terms, outcome, results, cid) 2g1

This procedure invokes, with arguments ARGS and modifiers MODS, the external procedure PNAME in package PKID in the environment known to the host environment via VID. Invoked locally (without the modifier), $CALL is the host environment's mechanism for calling remote procedures. Invoked remotely (with the modifier), it is the means by which procedure calls are relayed to environments known, but not adjacent in the environment tree, to the invoking environment. 2g2

DISP, TERMS, OUTCOME, RESULTS, and CID are explained in the PCP document. MODE specifies the mode -- either serial or parallel (with respect to continued operation of the host environment) -- in which execution of the remote procedure is to be carried out. 2g3

Argument/result types: 2g4

vid = INTEGER 2g4a
pkid = INTEGER 2g4b
pname = STRING 2g4c
args = any 2g4d
mods = any 2g4e
disp = LIST* (INTEGER [RETURN=0 / DISCARD=1] / dataref% DATAREF*) 2g4f
mode = BOOLEAN [SERIAL=TRUE / PARALLEL=FALSE] 2g4g
dummy = any 2g4h
terms = INTEGER [EXIT=0 / RPRT=1 / NOTE=2 / HELP=3 / WKNG=4] 2g4i
outcome= (outcome% / status% / event% / condition%) INTEGER / EMPTY 2g4j
results= (results% / intres% / eventdesc% / conddesc%) any / EMPTY 2g4k
cid = INTEGER / EMPTY 2g4l

Every environment is assumed to provide a primitive (probably not a procedure, and certainly not an external one) by which its procedures return to remote calling procedures. In its general form, this primitive might look like the following: 2g5

```
RETURN (terms, outcome, results : parms)                    2g5a
```

The following shorthands are also recommended: 2g6

```
Terminate a procedure normally                              2g6a
   $EXIT (outcome, results) ==>                             2g6a1
      $RETURN (EXIT, outcome, results)                      2g6a1a
Terminate a procedure abnormally                            2g6b
   $ABORT (errcode, errmsg) ==>                             2g6b1
      $RETURN (EXIT, ERROR, LIST (errcode, errmsg))         2g6b1a
Report intermediate results to caller                       2g6c
   $REPORT (status, intres : answer, newargs, newmods) ==>  2g6c1
      $RETURN (RPRT, status, intres : LIST (answer, newargs,
            newmods))                                       2g6c1a
Notify caller of an event                                   2g6d
   $NOTE (event, eventdesc) ==>                             2g6d1
      $RETURN (NOTE, event, eventdesc)                      2g6d1a
Solicit help from caller                                    2g6e
   $HELP (condition, conddesc : answer, helpfulparms) ==>   2g6e1
      $RETURN (HELP, condition, conddesc : LIST (answer,
            helpfulparms))                                  2g6e1a
```

Resume remote procedure 2h

$RESUME (pcid, parms, disp, mode -> terms, outcome, results, cid) 2h1

This procedure resumes with parameters PARMS, the remote procedure identified by PCID, 2h2

DISP, MODE, TERMS, OUTCOME, RESULTS, and CID (either EMPTY or equal in value to PCID) are as in $CALL. 2h3

Argument/result types: 2h4

pcid = INTEGER 2h4a
parms = any 2h4b
disp = LIST* (INTEGER [RETURN=0 / DISCARD=1] / dataref% DATAREF*) 2h4c
mode = BOOLEAN [SERIAL=TRUE / PARALLEL=FALSE] 2h4d
terms = INTEGER [EXIT=0 / RPRT=1 / NOTE=2 / HELP=3 / WKNG=4] 2h4e
outcome= (outcome% / status% / event% / condition%) INTEGER / EMPTY 2h4f
results= (results% / intres% / eventdesc% / conddesc%) any / EMPTY 2h4g
cid = INTEGER / EMPTY 2h4h

DATA STRUCTURES 3

$KWNEVM List of known environments 3a

This read-only data structure is a list of the names ENVNAMEs (of direct inferiors only) and handles VIDs for each of the environments presently known to the host environment (with the exceptions of SELF and SUPER). 3a1

Data structure type: 3a2

$kwnevm: LIST (envname: vid% INTEGER, ...) 3a2a

APPENDIX: PSEUDO IMPLEMENTATIONS 4

Introduction 4a

The following are implementations of some of SEVMCTL's procedures in something resembling SRI-ARC's L10 programming language. Their purpose is to help clarify the procedure definitions and to suggest, in broad terms, an implementation strategy. 4a1

Selected Procedures 4b

```
(Scrtevm) %create environment%                                          4b1
PROCEDURE (envname : vid)                                               4b1a
   %declarations%                                                       4b1a1
      LOCAL POINTER port;                                               4b1b
   %allocate and initialize port%                                       4b1b1
      initport ((port _ findport (status=free)),                        4b1b1a
         status    _ active,                                            4b1b1b
         type      _ inferior,                                          4b1b1c
         bckloc    _ self,                                              4b1b1d
         bckrem    _ self,                                              4b1b1e
         forloc    _ self,                                              4b1b1f
         forrem    _ self,                                              4b1b1g
         envname   _ envname,                                           4b1b1h
         compath   _ opnpth (envname),                                  4b1b1i
         entcnt    _ 0,                                                 4b1b1j
         wkng      _ false,                                             4b1b1k
         pklist    _ empty,                                             4b1b1l
         opnpklist _ empty);                                            4b1c
   %return%                                                             4b1c1
      $EXIT (success, port.vid);                                        4b1c2
      END.

(Sdelevm) %delete environment%                                          4b2
PROCEDURE (vid);                                                        4b2a
   %declarations%                                                       4b2a1
      LOCAL POINTER port, portp;                                        4b2b
   %locate and validate port%                                           4b2b1
      port _ findport (status=active & vid=vid &
       type=inferior);                                                  4b2b2
      IF port.entcnt > 0 THEN                                           4b2b2a
         $ABORT (code, "Control thread tangled.");                      4b2c
   %close port%                                                         4b2c1
      sreset () @ vid;                                                  4b2c2
      clspth (port.compath);                                            4b2d
   %discard broken links%                                               4b2d1
      WHILE (portp _ findport (status=active &
       type=head/link/tail & forloc/bckloc=vid)) DO                     4b2d1a
         IF portp.forloc = vid THEN portp.status _ dead                 4b2d1b
         ELSE srelhnd (portp.vid [portp.bckrem]);                       4b2e
   %release port%                                                       4b2e1
      port.status _ free;                                               4b2f
   %return%                                                             4b2f1
      $EXIT (success);                                                  4b2f2
      END.
```

```
($gethnd) %obtain environment handle%
PROCEDURE (vidlist [backlink] : headvid, tailvid);                    4b3
   %declarations%                                                     4b3a
      LOCAL listelm, tailvid;                                         4b3a1
      LOCAL POINTER port;                                             4b3a2
   %pop top element off of VID stack%                                 4b3b
      IF LENGTH (vidlist) = 0 THEN                                    4b3b1
         IF backlink # empty THEN listelm _ empty                     4b3b1a
         ELSE $ABORT (code, "Null VID list.")                         4b3b1b
      ELSE                                                            4b3b2
         BEGIN                                                        4b3b2a
         listelm _ vidlist (LENGTH (vidlist));                        4b3b2b
         BUMP DOWN LENGTH (vidlist);                                  4b3b2c
         port _ findport (status=active & vid=listelm);               4b3b2d
         IF backlink = empty AND port.type = link THEN                4b3b2e
            $ABORT (code, "Unassigned VID.");                         4b3b2e1
         END;                                                         4b3b2f
   %allocate and initialize link element%                             4b3c
      port _ findport (status=free);                                  4b3c1
      tailvid _ port.vid;                                             4b3c2
      initport (port,                                                 4b3c3
         status    _ active,                                          4b3c3a
         type      _ IF backlink = empty THEN head                    4b3c3b
            ELSE IF listelm = empty THEN tail ELSE link,              4b3c3b1
         bckloc    _ IF backlink = empty THEN self                    4b3c3c
            ELSE masters [LENGTH (masters)],                          4b3c3c1
         bckrem    _ IF backlink = empty THEN self ELSE
          backlink,                                                   4b3c3d
         forloc    _ IF listelm = empty THEN self ELSE listelm,       4b3c3e
         forrem    _ IF listelm = empty THEN self ELSE                4b3c3f
            $gethnd (vidlist [port.vid] ; tailvid) @ listelm,         4b3c3f1
         envname   _ "",                                            4b3c3g
         compath   _ empty,                                         4b3c3h
         entcnt    _ 0,                                             4b3c3i
         wkng      _ false,                                         4b3c3j
         pklist    _ empty,                                         4b3c3k
         opnpklist _ empty);                                        4b3c3l
   %return%                                                           4b3d
      $EXIT (success, portp.vid, tailvid);                            4b3d1
      END.                                                            4b3d2
```

```
(srelhnd) %release environment handle%                                  4b4
PROCEDURE (vid [backlink]);                                             4b4a
   %declarations%                                                       4b4a1
      LOCAL POINTER port, portp;                                        4b4b
   %locate chain element%                                               4b4b1
      port _ findport (status=active/dead & vid=vid &
       type=head/link/tail);                                            4b4b2
      IF backlink = empty AND port.type # head THEN                     4b4b2a
         $ABORT (code, "Unassigned VID,");                              4b4b3
      IF port.entCnt > 0 THEN                                           4b4b3a
         $ABORT (code, "Control thread tangled,");                      4b4c
   %discard broken links%                                               4b4c1
      IF port.type # link THEN                                          4b4c1a
         WHILE (portp _ findport (status=active &
          type=head/link/tail & forloc/bckloc=port,vid)) DO             4b4c1a1
            IF portp.forloc = port,vid THEN portp.status _ dead         4b4c1a2
            ELSE $relhnd (portp,vid [portp.bckrem]);                    4b4d
   %delete link%                                                        4b4d1
      IF port.type # tail THEN                                          4b4d1a
         $relhnd (port.forrem, port.vid) @ port.forloc;                 4b4d2
      port.status _ free;                                               4b4e
   %return%                                                             4b4e1
      $EXIT (success);                                                  4b4e2
      END.
```

```
(scall) %call remote procedure%
PROCEDURE (vid, pkid, pname, args, mods, disp, mode [dummy] :
 terms, outcome, results, cid);                                          4b5
  %declarations%                                                         4b5a
    LOCAL INTEGER terms, outcome, cid, dir, locvid, remvid,
     i, addr, boss;                                                      4b5a1
    LOCAL LIST state, results;                                           4b5a2
    LOCAL POINTER port, call;                                            4b5a3
  %follow link if any%                                                    4b5b
    dir _ cid _ empty;                                                   4b5b1
    boss _ masters [LENGTH (masters)];                                   4b5b2
    port _ findport (status=active & vid=vid);                           4b5b3
    CASE port.type OF                                                    4b5b4
      =head:                                                            4b5b4a
        IF port.forloc # boss THEN                                     4b5b4a1
          BEGIN                                                       4b5b4a1a
          locvid _ port.forloc;                                       4b5b4a1b
          remvid _ port.forrem;                                       4b5b4a1c
          dir _ forward;                                              4b5b4a1d
          END                                                         4b5b4a1e
        ELSE locvid _ self;                                            4b5b4a2
      =tail:                                                            4b5b4b
        IF port.bckloc # boss THEN                                     4b5b4b1
          BEGIN                                                       4b5b4b1a
          locvid _ port.bckloc;                                       4b5b4b1b
          remvid _ port.bckrem;                                       4b5b4b1c
          dir _ backward;                                             4b5b4b1d
          END                                                         4b5b4b1e
        ELSE locvid _ self;                                            4b5b4b2
      =link:                                                            4b5b4c
        IF dummy = empty OR port.bckloc&forloc # boss THEN            4b5b4c1
          $ABORT (code, "Unassigned VID.")                            4b5b4c1a
        ELSE IF port.bckloc = boss                                     4b5b4c2
          THEN REPEAT CASE head                                       4b5b4c2a
          ELSE REPEAT CASE tail;                                      4b5b4c2b
      ENDCASE %self/superior/inferior%                                  4b5b4d
        locvid _ port.vid;                                             4b5b4d1
  %distant environment%                                                   4b5c
    IF dir # empty THEN                                                  4b5c1
      terms _                                                           4b5c1a
        $call (remvid, pkid, pname, args, mods, disp, mode
         [dummy] : outcome, results, cid) @ locvid                    4b5c1a1
  %adjacent environment%                                                  4b5d
    ELSE IF locvid # self THEN                                           4b5d1
      BEGIN                                                             4b5d1a
      %verify environment idle%                                         4b5d1b
```

```
            IF port.wkng THEN                                          4b5d1b1
               $ABORT (code, "Environment busy,");                     4b5d1b1a
            %send call message%                                         4b5d1c
               sndpth (port.compath, LIST (call, pkid, pname,
                args, mods, disp));                                    4b5d1c1
               BUMP port.entcnt;                                       4b5d1c2
            %accept return message%                                     4b5d1d
               IF mode = serial THEN                                   4b5d1d1
                  WHILE (OUTCOME OF (terms _ getrtn (port.compath
                   : outcome, results, cid)) # success) DO NULL        4b5d1d1a
               ELSE                                                    4b5d1d2
                  BEGIN                                                4b5d1d2a
                  port.wkng _ TRUE;                                    4b5d1d2b
                  terms _ wkng;                                        4b5d1d2c
                  outcome _ results _ empty;                           4b5d1d2d
                  END;                                                 4b5d1d2e
            END                                                         4b5d1e
         %local environment%                                            4b5e
            ELSE                                                        4b5e1
               BEGIN                                                    4b5e1a
               %force serial operation%                                 4b5e1b
                  IF mode # serial THEN                                 4b5e1b1
                     $ABORT (code, "Run self serially only,");          4b5e1b1a
               %verify open package%                                    4b5e1c
                  LOOP                                                  4b5e1c1
                     BEGIN                                              4b5e1c1a
                     FOR i_1 TO LENGTH (port.opnpklist) DO              4b5e1c1b
                        IF port.opnpklist [i] = pkid THEN EXIT LOOP
                         2;                                             4b5e1c1b1
                     $ABORT (code, "Undefined PKID,");                  4b5e1c1c
                     END;                                               4b5e1c1d
               %make local procedure call%                              4b5e1d
                  addr _ getaddr (pkid, pname);                         4b5e1d1
                  terms _ addr (args, mods : terms, outcome, results,
                   state);                                              4b5e1d2
                  results _ dspres (results, disp);                     4b5e1d3
               END;                                                     4b5e1e
         %assign CID%                                                   4b5f
            IF terms # exit THEN                                        4b5f1
               BEGIN                                                    4b5f1a
               initcall ((call _ findcall (status=free)),               4b5f1b
                  status _ active,                                      4b5f1b1
                  type   _ IF dummy = empty THEN head ELSE link,        4b5f1b2
                  forvid _ locvid,                                      4b5f1b3
                  forcid _ cid,                                         4b5f1b4
                  terms  _ terms,                                       4b5f1b5
```

```
        state _ IF locvid = self THEN state ELSE empty);   4b5f1b6
      cid _ call.cid;                                      4b5f1c
      END;                                                 4b5f1d
%return%                                                   4b5g
   $RETURN (terms, outcome, results, cid);                 4b5g1
   END.                                                    4b5g2

                                                           4b5g3
```

```
(sresume) %resume remote procedure%
PROCEDURE (pcid, parms, disp, mode : terms, outcome, results,
 cid);                                                          4b6
   %declarations%                                               4b6a
      LOCAL INTEGER terms, outcome, cid, locvid, boss;          4b6a1
      LOCAL LIST state, results;                                4b6a2
      LOCAL POINTER port, call;                                 4b6a3
   %locate control thread%                                      4b6b
      call _ findcall (status=active & cid=pcid);               4b6b1
      IF dummy = empty AND call.type # head THEN                4b6b2
         sABORT (code, "Unassigned CID.");                      4b6b2a
      port _ findport (status=active & vid=call.forvid);        4b6b3
   %follow link if any%                                         4b6c
      boss _ masters [LENGTH (masters)];                        4b6c1
      locvid _ CASE port.type OF                                4b6c2
         =head:                                                 4b6c2a
            IF port.forloc # boss THEN port.forloc ELSE self;   4b6c2a1
         =tail:                                                 4b6c2b
            IF port.bckloc # boss THEN port.bckloc ELSE self;   4b6c2b1
         =link:                                                 4b6c2c
            IF port.bckloc = boss                               4b6c2c1
               THEN REPEAT CASE head                            4b6c2c1a
               ELSE REPEAT CASE tail;                           4b6c2c1b
         ENDCASE %self/superior/inferior% port.vid;             4b6c2d
   %local environment%                                          4b6d
      IF locvid = self THEN                                     4b6d1
         IF mode # serial THEN                                  4b6d1a
            sABORT (code, "Resume self serially only.")         4b6d1a1
         ELSE                                                   4b6d1b
            BEGIN                                               4b6d1b1
            terms _ resume (call.state, parms : outcome,
             results, state)                                    4b6d1b2
            results _ dspres (results, disp);                   4b6d1b3
            END                                                 4b6d1b4
   %remote environment%                                         4b6e
      ELSE                                                      4b6e1
         BEGIN                                                  4b6e1a
         %propagate resumption%                                 4b6e1b
            IF call.terms # wkng THEN                           4b6e1b1
               sndpth (port.compath, LIST (resume, call.forcid,
                parms, disp));                                  4b6e1b1a
         %serial resumption%                                    4b6e1c
            IF mode = serial THEN                               4b6e1c1
               WHILE (OUTCOME OF (terms _ getrtn (port.compath
                : outcome, results)) # success) DO NULL         4b6e1c1a
         %parallel resumption%                                  4b6e1d
```

```
            ELSE                                                    4b6e1d1
               IF OUTCOME OF tstpth (port,compath) = success        4b6e1d1a
               AND OUTCOME OF (terms _ getrtn (port,compath :
                outcome, results)) = success THEN                   4b6e1d1b
                  port.wkng _ FALSE                                 4b6e1d1b1
               ELSE terms _ wkng                                    4b6e1d1c
         END;                                                       4b6e1e
%release CID%                                                       4b6f
      IF terms = exit THEN                                          4b6f1
         BEGIN                                                      4b6f1a
         BUMP DOWN port.entcnt;                                     4b6f1b
         call.status _ free;                                        4b6f1c
         cid _ empty;                                               4b6f1d
         END                                                        4b6f1e
      ELSE                                                          4b6f2
         BEGIN                                                      4b6f2a
         call.terms _ terms;                                        4b6f2b
         call.state _ state;                                        4b6f2c
         cid _ call.cid;                                            4b6f2d
         END;                                                       4b6f2e
%return%                                                            4b6g
      $RETURN (terms, outcome, results, cid);                       4b6g1
      END.                                                          4b6g2
```

Subroutines 4c

```
(getrtn) %accept return from adjacent environment%                    4c1
PROCEDURE (port : terms, outcome, results, cid);                      4c1a
   %declarations%                                                     4c1a1
      LOCAL INTEGER terms, outcome, cid, op, pkid, pcid,
       answer;                                                       4c1a2
      LOCAL STRING pname;                                           4c1a3
      LOCAL LIST parms;                                             4c1a4
      LOCAL WILD results, args, mods, disp;                          4c1b
   %fetch message from adjacent environment%                        4c1b1
      LIST (op, pkid, pname, args, mods, disp) / LIST (op,
       pcid, parms, disp) / LIST (op, terms, outcome, results,
       cid) _ rcvpth (port,compath);                                4c1c
   %return with procedure return%                                   4c1c1
      IF op = return THEN                                          4c1c1a
         BEGIN                                                     4c1c1b
         IF terms = exit THEN BUMP DOWN port.entcnt;               4c1c1c
         port.wkng _ FALSE;                                        4c1c1d
         $EXIT (success, terms, outcome, results, cid);            4c1c1e
         END;                                                       4c1d
   %field call/resumption of local procedure%                       4c1d1
      masters [BUMP LENGTH (masters)] _ port.vid;                   4c1d2
      terms _ IF op = call THEN
         THEN $call (self, pkid, pname, args, mods, disp,
          serial : outcome, results, cid)                          4c1d2a
         ELSE $resume (pcid, parms, disp, serial : outcome,
          results, cid);                                           4c1d2b
      BUMP DOWN LENGTH (masters);                                   4c1d3
      sndpth (port,compath, LIST (return, terms, outcome,
       results, cid));                                              4c1d4
   %return%                                                         4c1e
      $EXIT (failure);                                              4c1e1
      END.                                                          4c1e2

                                                                    4c1e3
```

```
(dspres) %dispose of procedure results%                                  4c2
PROCEDURE (rawres, disp : results);                                      4c2a
   %declarations%                                                        4c2a1
      LOCAL INTEGER i;                                                   4c2b
   %individualized disposition%                                          4c2b1
      IF TYPE disp = list THEN                                           4c2b1a
         FOR i_1 TO LENGTH (rawres) DO                                   4c2b1a1
            CASE disp [i] OF                                             4c2b1a1a
               =return: NULL;                                            4c2b1a1b
               =discard: rawres [i] _ empty;                             4c2b1a1c
               ENDCASE                                                   4c2b1a1c1
                  BEGIN                                                  4c2b1a1c2
                  $wrdata (disp [i], rawres [i]);                        4c2b1a1c3
                  REPEAT CASE discard;                                   4c2b1a1c4
                  END
   %blanket disposition%                                                 4c2c
      ELSE                                                               4c2c1
         CASE disp OF                                                    4c2c1a
            =return: NULL;                                               4c2c1a1
            =discard: rawres _ empty;                                    4c2c1a2
            ENDCASE                                                      4c2c1a3
               BEGIN                                                     4c2c1a3a
               $wrdata (disp, rawres [LENGTH (rawres)]);                 4c2c1a3b
               REPEAT CASE discard;                                      4c2c1a3c
               END;                                                      4c2c1a3d
   %return%                                                              4c2d
      $EXIT (success, rawres);                                           4c2d1
      END.                                                               4c2d2
```

Internal Global Data Structures 4d

GLOBAL RECORD port 4d1

```
INTEGER vid,        %VID to which this port corresponds%          4d1a
INTEGER status,     %status of this VID%                          4d1b

   %free   -- no environment associated with VID%                 4d1b1
   %active -- environment assigned to VID%                        4d1b2
   %dead   -- environment assigned to VID but it's dead%          4d1b3

INTEGER type,       %relationship of remote environment%          4d1c

   %self     -- the host environment%                             4d1c1
   %superior -- host environment's direct superior%               4d1c2
   %inferior -- a direct inferior%                                4d1c3
   %head     -- a VID obtained (as headvid) via $GETHND%          4d1c4
   %link     -- support for $GETHND VIDs known elsewhere%         4d1c5
   %tail     -- a VID obtained (as tailvid) via $GETHND%          4d1c6

INTEGER bcklOc,     %VID of previous environment in chain%        4d1d
   %self -- if this is the head%                                  4d1d1
INTEGER bckrem,     %VID of link element in previous
 environment%                                                     4d1e
   %self -- if this is the head%                                  4d1e1
INTEGER forloc,     %VID of next environment in chain%            4d1f
   %self -- if this is the tail%                                  4d1f1
INTEGER forrem,     %VID of link element in next environment%     4d1g
   %self -- if this is the tail%                                  4d1g1
STRING  envname,    %environment name%                            4d1h
   %null -- unless direct inferior%                               4d1h1
INTEGER compath,    %IEC path handle%                             4d1i
   %null -- (for now) unless direct inferior/superior%           4d1i1
INTEGER entcnt,     %# sent CALLs still awaiting EXITs%           4d1j
BOOLEAN wkng,       %TRUE if sent CALL still awaiting RETURN%     4d1k
LIST    pklist,     %PKIDs for accessible local packages%         4d1l
LIST    opnpklist;  %PKIDs for open local packages%               4d1m
```

```
GLOBAL RECORD call                                                    4d2

   INTEGER cid,    %CID to which this call element corresponds%       4d2a
   INTEGER status, %status of this CID%                               4d2b

      %free   -- no call associated with CID%                         4d2b1
      %active -- call assigned to CID%                                4d2b2

   INTEGER type,   %type of element%                                  4d2c

      %head -- CID returned to calling procedure%                     4d2c1
      %link -- support for CID returned elsewhere                     4d2c2

   INTEGER forvid, %VID of environment with next element in
                    link%                                             4d2d
   INTEGER forcid, %CID of call element in next environment%          4d2e
   INTEGER terms,  %TERMS with which procedure RETURNed%              4d2f
   LIST    state;  %procedure state record%                           4d2g

GLOBAL RECORD pack                                                    4d3

   INTEGER pkid,    %PKID to which this package record
   corresponds%                                                       4d3a
   STRING  pkname,  %package name%                                    4d3b
   INTEGER opncnt;  %number of environments with package open%        4d3c

GLOBAL LIST masters;  %VIDs of calling environments%                  4d4
   %top element -- VID of controlling environment%                    4d4a
```

DRAFT $EVMCTL
The Environmental Control Package

3-SEP-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

$EVMCTL is an organizational tool that operates within the setting provided by the Procedure Call Protocol (PCP -- xxxxxx,), with which the reader of the present document is assumed familiar.

DRAFT JEW 2 SEP 74 7:41PM The Low-Level Debug Package

(J23907) 2-SEP-74 04:35;;;; Title: Author(s): James E. (Jim) White/JEW; Distribution: /NPG([INFO-ONLY]) JBP([INFO-ONLY]) RWW([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: JEW; Origin: < WHITE, PCP-LLDBUG.NLS;6, >, 2-SEP-74 01:18 JEW ;;;; ####;

One of a series of related documents,

INTRODUCTION 1

The Low-Level Debug Package (package name=$LLDBUG) contains those procedures and data structures which a remote environment requires to debug the host environment at the assembly-language level. The package contains procedures for manipulating and searching the host environment's address space, for manipulating and searching its symbol tables, and for setting and removing breakpoints from the environment's address space. It contains data structures which hold environment characteristics and state information, and the contents of program symbol tables. 1a

This package is appropriately supplied with any environment that can be usefully modeled as data and executable code, resident in a single address space (consisting of an ordered set of words, addressed 0 through n), and executed by means of a single program counter (PC). 1b

Throughout this document, the following shorthands denote, respectively, a program symbol, and an address in either absolute or symbolic form: 1c

```
SYMBOL*  ==> tblname: symname% STRING                          1c1
ADDRESS* ==> INTEGER / LIST (SYMBOL*, offset% INTEGER)         1c2
```

PROCEDURES 2

The Address Space 2a

Read address space 2a1

$RDCORE (strtaddr, wrdcnt, encoding => values) 2a1a

This procedure retrieves from the host environment's address space, the current contents VALUES of the contiguous block of WRDCNT words beginning at address STRTADDR. ENCODING specifies the manner in which the contents of each word are to be encoded for return: 2a1b

TXT: as text (result type = STRING) 2a1b1
CDE: as an executable instruction (result type = STRING) 2a1b2
INT: as a signed integer (result type = INTEGER) 2a1b3
WRD: uninterpreted (result type = WORD) 2a1b4

Argument/result types: 2a1c

strtaddr- ADDRESS* 2a1c1
wrdcnt - INTEGER 2a1c2
encoding- INTEGER [TXT=0 / CDE=1 / INT=2 / WRD=3] 2a1c3
values - LIST (STRING / INTEGER / WORD, ...) 2a1c4

Write address space 2a2

$WRCORE (strtaddr, wrdcnt, values, encoding) 2a2a

This procedure replaces the current contents of the contiguous block of WRDCNT words beginning at address STRTADDR in the host environment's address space, with the new values VALUES. ENCODING specifies the manner in which the new contents of each word have been encoded by the invoking environment (same as in $RDCORE). 2a2b

Argument/result types: 2a2c

strtaddr- ADDRESS* 2a2c1
wrdcnt - INTEGER 2a2c2
values - LIST (STRING / INTEGER / WORD, ...) 2a2c3
encoding- INTEGER [TXT=0 / CDE=1 / INT=2 / WRD=3] 2a2c4

Search address space 2a3

$SEARCH (strtaddr, wrdcnt, value, encoding, mask -> addrs) 2a3a

This procedure searches the contiguous block of WRDCNT words beginning at address STRTADDR in the host environment's address space, for those words ADDRS whose content matches VALUE, after both have been ANDed with the mask MASK. ENCODING specifies the manner in which the comparand VALUE has been encoded by the invoking environment (same as in $WRCORE). 2a3b

Argument/result types: 2a3c

```
strtaddr= ADDRESS*                                          2a3c1
wrdcnt  = INTEGER                                           2a3c2
value   = STRING / INTEGER / WORD                           2a3c3
encoding= INTEGER [TXT=0 / CDE=1 / INT=2 / WRD=3]           2a3c4
mask    = WORD                                              2a3c5
addrs   = LIST (ADDRESS*, ...)                              2a3c6
```

Symbol Tables 2b

Open symbol table 2b1

sOPSYMT (tblname) 2b1a

This procedure opens the host environment's symbol table TBLNAME. 2b1b

Argument/result types: 2b1c

tblname- STRING 2b1c1

Close symbol table 2b2

sCLSYMT (tblname) 2b2a

This procedure closes the host environment's previously-opened symbol table TBLNAME. 2b2b

Argument/result types: 2b2c

tblname- STRING 2b2c1

Create symbol 2b3

sCRTSYM (symbol, value) 2b3a

This procedure adds the symbol SYMBOL with value VALUE to one of the host environment's previously-opened symbol tables (implicitly named by SYMBOL). 2b3b

Argument/result types: 2b3c

symbol- SYMBOL* 2b3c1
value - ADDRESS* 2b3c2

Delete symbol 2b4

sDELSYM (symbol) 2b4a

This procedure deletes the symbol SYMBOL from one of the host environment's previously-opened symbol tables (implicitly named by SYMBOL). 2b4b

Argument/result types: 2b4c

symbol- SYMBOL* 2b4c1

Read symbol value 2b5

$RDSYM (symbol -> value) 2b5a

This procedure returns the value VALUE of the symbol SYMBOL in one of the host environment's previously-opened symbol tables (implicitly named by SYMBOL). 2b5b

Argument/result types: 2b5c

symbol- SYMBOL* 2b5c1
value - INTEGER 2b5c2

Write symbol value 2b6

$WRSYM (symbol, value) 2b6a

This procedure assigns the value VALUE to the symbol SYMBOL in one of the host environment's previously-opened symbol tables (implicitly named by SYMBOL). 2b6b

Argument/result types: 2b6c

symbol- SYMBOL* 2b6c1
value - ADDRESS* 2b6c2

Fit value to symbol table 2b7

$FTVAL (comparand, tblname -> symbol, value) 2b7a

This procedure returns the name SYMBOL and value VALUE of the symbol, in the host environment's previously-opened symbol table TBLNAME (or in any of its symbol tables, if TBLNAME is EMPTY), whose current value is closest to COMPARAND. 2b7b

Argument/result types: 2b7c

comparand- ADDRESS* 2b7c1
tblname - STRING / EMPTY 2b7c2
symbol - SYMBOL* 2b7c3
value - INTEGER 2b7c4

Breakpoints 2c

Create breakpoint 2c1

$SETBRK (addr, pcdcnt) 2c1a

This procedure sets a breakpoint at address ADDR in the host environment's address space. The PCDCNTth time the environment's PC reaches the breakpoint, the environment will be "frozen" (i.e. its state will be stored in $EVMSTA), the primitive $NOTE (BRKPNT, addr) will be invoked, and then the environment will be "thawed" (i.e. its state will be restored from $EVMSTA). 2c1b

Argument/result types: 2c1c

addr = ADDRESS* 2c1c1
pcdcnt= INTEGER 2c1c2

Delete breakpoint(s) 2c2

$REMBRK (addr) 2c2a

This procedure removes the breakpoint previously set at address ADDR in the host environment's address space or, if ADDR is EMPTY, removes all breakpoints from its address space (as does closing $LLDBUG). 2c2b

Argument/result types: 2c2c

addr= ADDRESS* / EMPTY 2c2c1

Execute intruction 2c3

$EX (inst, encoding) 2c3a

This procedure thaws the host environment, executes the single instruction INST, and then re-freezes the environment. ENCODING specifies the manner in which INST has been encoded by the invoking environment (same as in $WRCORE). 2c3b

Argument/result types: 2c3c

inst = STRING / INTEGER / WORD 2c3c1
encoding= INTEGER [TXT=0 / CDE=1 / INT=2 / WRD=3] 2c3c2

DATA STRUCTURES 3

$EVMSTA Environment state 3a

This data structure holds the host environment's state. The environment's state is saved in $EVMSTA: 3a1

1) whenever $LLDBUG is opened, 3a1a
2) whenever the environment reaches a breakpoint, and 3a1b
3) after an instruction has been executed by means of $EX. 3a1c

and restored from $EVMSTA: 3a2

1) whenever $LLDBUG is closed, 3a2a
2) whenever the environment is continued from a breakpoint (i.e. when $NOTE returns control), 3a2b
3) before an instruction is executed by means of $EX. 3a2c

If the host environment supports the Environment Control package, described in another document, $EVMSTA represents the environment state (if any) most recently saved by means of the $INTEVM procedure. 3a3

$EVMSTA is somewhat environment-dependent in format and content, but always contains at least the contents of the environment's program counter PC and its general registers REGS (if any). 3a4

Data structure type: 3a5

```
sevmsta: LIST (pc: ADDRESS*, regs: LIST* (WORD, ...), any,
         ...)
```
3a5a

$EVMCHR Environment characteristics 3b

This read-only data structure contains certain characteristic information about the host environment. 3b1

$EVMCHR is somewhat environment-dependent in format and content, but always contains at least the number of words ASIZE in the environment's address space, and the width WRDLEN in bits of each word. 3b2

Data structure type: 3b3

```
sevmchr: LIST (asize: INTEGER, wrdlen: INTEGER, any, ...)
```
3b3a

$SYMTBS Symbol tables 3c

This read-only data structure represents the contents of all of the host environment's open symbol tables, and contains the name SYMBOL and value VALUE of each symbol in each open table TBLNAME. 3c1

Data structure type: 3c2

ssymtbs: LIST (tblname: LIST (symbol: value% INTEGER, ...), ...) 3c2a

JEW 2-SEP-74 04:35 23907

DRAFT $LLDBUG
The Low-Level Debug Package

3-SEP-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

$LLDBUG is a debugging tool that operates within the setting provided by the Procedure Call Protocol (PCP -- xxxxxx,), with which the reader of the present document is assumed familiar.

SRL 3-SEP-74 07:52 23908

Comments on draft (23886,)

(J23908) 3-SEP-74 07:52;;;; Title: Author(s): Susan R. Lee/SRL;
Distribution: /DVN([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk:
SRL;

SRL 3-SEP-74 07:52 23908

Comments on draft (23886,)

I just finished reading your draft on transisting from old to new nls (23886,). My only thought was that the section on recognition should precede the section on question mark since it's discussed under question mark. Also, I was reading quickly cbut don't remember anything about changes in TNLS like & = isn't jump to file return something different? Looks good and should be helpful. Heard you all had a good Chinese banquet - hope to be back in CA one of these months!

1

DVN 3-SEP-74 09:00 23909

My Vacation Time

(J23909) 3-SEP-74 09:00;;;; Title: Author(s): Dirk H. Van Nouhuys/DVN; Distribution: /RWW([ACTION]) SLJ([INFO-ONLY]) JOAN([INFO-ONLY]) JMB([INFO-ONLY]) KIRK([INFO-ONLY]) JCN([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: DVN;

DVN 3-SEP-74 09:00 23909

My Vacation Time

I'm scheduled to be on vacation next week, Both for geting out the documentation and personal convenience it would be better for me to be away the following week (Sept16-20) and unless some one sees problems I will do so,

1

INTRODUCTION

NLS or online system is the name of the computer system you will be using. Online means you receive immediate feedback about what you have just typed at your terminal.

NLS has facilities to let you do almost everything you need with text: compose it; edit it; send it to (and receive it from) other persons; file it in one or more categories; cite and easily obtain documents; search for documents by author and subject; search in documents by word or phrase; and print in practically any format.

This primer demonstrates the commands used for writing a memo, editing it, and distributing it to other people. This process is explained for TNLS which is the typewriter version of NLS. You will find it useful to be at a terminal, typing in the commands and text as the primer describes them.

Although this primer describes specific functions, we add notes at each step which generalize the operation. Given this primer as a model, the inexperienced user should be able to perform any of the operations described here and refer to the full NLS documentation for more information about the system.

Throughout this primer, we spell out the sequence of keys you strike to make something happen and separately show what will appear on your terminal in response. Keys that do not print, such as carriage return, altmode (called "escape" on some terminals), and control characters, are named inside angle brackets, e.g. <CR>, <ALT>, and <CTRL-C>. <> represents a space. Information printed by the system is shown in a special typeface. The control key (CTRL) is used like the shift key. You hold it down while you type the letter that is after the hyphen.

OOPS...If you type an incorrect letter or number, just type a backspace or <CTRL-A> immediately following your error and then you can type the correct character.

Are you stuck?? Confused?? Don't know what to type next??
Typing "?" will show you the next possible alternatives.
Typing <CR> will put you where you were before you typed "?."
Typing <CTRL-Q> will provide you with information and explanations about NLS.
Typing <CTRL-X> will put you where you were before you typed <CTRL-Q>.
For more about getting information via <CTRL-Q> see the last section of this primer.

What is the meaning of <CR>?
A. County Registrar
B. Cute Reindeer
C. Carriage Return

The correct choice is C. When you see <CR>, use the return or carriage return on your keyboard.

INSTRUCTION

Most users of this primer will reach NLS through the ARPA Network. For the current connection procedures at your site, see some one knowledgeable in NLS. When you have made your connection you will see "TENEX 1.##.## SITENAME 1.##.##" which is called a "header" and an "@" which is a signal called a herald. The herald tells you that TENEX, (a system within the computer that assigns service to users) is waiting for you, the user, to identify yourself.

1. To identify yourself to the TENEX system at Office-1,

If you type,	you should see:
login<CR>	@login
DIRECTORYNAME<CR>	(USER)DIRECTORYNAME
PSWD<CR>	(PASSWORD)
<CR>	(ACCOUNT #)
	JOB # ON TTY # DATE TIME

If you do not know a DIRECTORYNAME or Password, ask the person in your organization who usually helps people with NLS or call (415) 326-6200 extension 3630 between 8am and 5pm pacific time.

2. To enter the TNLS system:

If you type,	you should see:
nls<CR>	@nls
	BASE C:

When you enter NLS, it prints "Base" which is the herald of its central subsystem called Base. In using <CTRL-A> to correct a mistake, when you see only "Base C:", you may begin again.

3. Since you are going to write a memo, you will need an empty file (or workspace) in which to put it. You give the file a name so that you can call it back in future NLS sessions.

If you type,	you should see:
<>crfmemo<CR>	BASE C: Create C:File T:memo < DIRECTORYNAME, MEMO,NLS;1 > BASE C:

Where NLS expects to do something, it asks you for a command word by prompting you with a C: and where it expects you to type in some text, it prompts you with T:.

The system usually finishes or completes a command word after you have typed in the first letter. In the case of some comands used less often, you have to type a space and then one, two, or three letters. This is called Terse recognition. NLS offers other modes of recognition. To find out about them, type <CTRL-Q> and then the words "command recognition."

You now have a new and empty file named MEMO. Filenames may be up to 29 letters and digits beginning with a letter. File names may not include spaces, commas, or periods.

If you leave the system without finishing your work, you can retrieve it (or any other stored file) in TNLS by using the command, Load File. YOU DO NOT NEED TO DO THIS NOW, because your file named MEMO is already loaded.

If you type,	you should see:
lfmemo<CR>	BASE C:Load C: File T: memo BASE C:

4. Now that you have created MEMO, the system has already inserted some information at the file's beginning or at the statement numbered 0. Statement 0 identifies MEMO to NLS and is generally unused by you except to cite the beginning of the file. To see the statement you are currently at, i.e., statement 0, type: \

.....

The response will be:

```
BASE C:\
< DIRCTORYNAME,MEMO,NLS;1 >,    DATE    TIME    IDENT ;;;;
BASE C:
```

.....

5. You begin writing your memo by indicating you are going to insert a statement into your file MEMO starting after statement 0, and then by actually typing in some text. Statements are comparable to paragraphs of text. The system will automatically move the print head back when it runs out of room at the end of a line. You do not need a carriage return at the end of each line. The lines may not be the same as in the examples. Note intentional typogoofical errors for future correction.

.....

If you type,	you should see:
is0<CR>	BASE C: Insert C: Statement (to follow)A:0
<CR>	L:
Contradictions have	T:Contradictions have been
been alledged in our	alledged in our description
description of the	of the elephant.
elephant.<CR>	BASE C:

.....

Notice you are prompted for specific types of input, in this case A: asks you for an address, T: for typein. An address specifies a point in a file. In this case, you gave an address of "after statement 0" because that was where you wanted your new statement to begin. If you were creating a file that used an outline structure, L: would prompt you to specify the level in the outine where you wanted to put each statement. In this primer you can ignore L: by typing a <CR>.

After this command is executed, the statement "Contradictions have been..." is inserted after statement 0, i.e., at the beginning of the file, and assigned the statement number 1.

6. Since statement numbers are invaluable for keeping track of what statements are where, you will want to see them as you work on your file.

If you type,	you should see:
<>sevm<CR>	BASE C: set C: Viewspecs V:m BASE C:

This command accepts codes that control the "view" you have of your memo; m makes the system number statements in printing.

7. As you enter statements into the file, you will periodically want to check how the memo looks as you go along. You can look at all or part of your file by printing it. To see only the statement you are at currently, type: \

The response will be:

```
BASE C:\
1  Contradictions have been alledged in our description of
the elephant.
BASE C:
```

Later on when there are more statements in your file you can see more by using the Print Rest command, described in step 14.

8. Step 5 showed you how to enter one statement, more commonly, you will want to enter several statements, one after the other. Instead of repeating the Insert Statement command for each new statement, type the character <CTRL-E> at the end of your first statement. This tells the system to continue the Insert Statement command. We call this repeat insert, or insert mode. Once you get in the insert mode, you end each statement you type in with a <CR>, and then immediately type in another statement. Follow your last statement with a <CR> and a <CTRL-X>. This will take you out of the insert mode. To add (after statement 1) three more statements to your file, completing the rough draft of your memo:

If you type,	you should see:
is1<CR>	BASE C:Insert C:Statement(to follow) A:1
<CR>	L:
The review meeting will be	T:The review meeting will be
at 3:00<CTRL-e>	at 3:00<^E>
<CR>	L:
Only wise, blind men	T:Only wise, blind men should attend.
should attend.<CR>	
<CR>	L:
A recurcive redefinition	T:A recurcive redefinition
plan should imerge.<CR>	plan should imerge.
<CTRL-X>	L:
	BASE C:

9. you have now completed a rough draft of your memo and want to check it for completeness, typing errors, etc. To review the content of the file you use the Print Rest command. The Print Rest command shown in Step 10 starts printing from the current statement to the end of the file, so you should first return to the beginning of the file before you use it. (Other versions of the Print command are described below). The command for going to the first statement you wrote (statement 1) is:

If you type,	you should see:
ja1<CR>	BASE C: Jump (to) C: Address A:1
	BASE C:

10. Use the Print Rest command to print the content of your memo from where you are to the end of your file.

If you type,	you should see:
pr<CR>	BASE C: Print C:Rest OK: 1 Contradictions have been alledged in our description of the elephant. 2 The review meeting will be at 3:00 3 Only wise, blind men should attend. 4 A recurcive redefinition plan should imerge. BASE C:

11. Now you might decide that statement 3 is superfluous. To delete statement 3:

If you type,	you should see:
ds3<CR> <CR>	BASE C: Delete C: Statement (at) A:3 OK: BASE C:

12. you may also decide to add text to the end of statement 2. To do so you use a command virtually identical to the insert statement command.

If you type,	you should see:
it2<>+e<CR> <>in the project room,<CR>	BASE C: Insert C: Text (to follow) A:2 +e T: in the project room. BASE C:

The significant difference in this command from the version you used to insert statements is that you specify where in the statement you want the text to go. The space followed by "+e" after the statement number tells the system to insert the text at the end of that statement.

"+e" is a convenient way to point to the end of a statement. However, if you want to insert text elsewhere in the statement you must specify exactly where. The easiest way to do so is to cite the place of insertion by content. Thus instead of using "+e" you might have specified ""3:00"" with identical results. The double pairs of quotation marks indicate that you use quotes when you specify content. Note that the specific within-statement location follows the statement number and is separated from it by a space. TNLS "reads" addresses from left to right.

Note also that the primer asks you to type a a space at the beginning of the insertion; that space avoids having "...3:00in the..." appear in the file.

13. If you strike \ you can look at statement 2 to check the changes.

The response should look like:

```
BASE C:\
2 The review meeting will be at 3:00 in the project room.
BASE C:
```

14. At this point you are ready to check your file for minor errors. Print it again as you did in Steps 9 and 10:

If you type,	you should see:
ja1<CR>	BASE C: Jump (to) C: Address A:1
pr<CR>	BASE C: Print C: Rest OK: 1 Contradictions have been alledged in our description of the elephant. 2 The review meeting will be at 3:00 in the project room. 3 A recurcive redefinition plan should imerge. BASE C:

Note that when you deleted the old statment 3, the system renumbered the remaining statements.

15. The most convenient way to correct the kind of typographical errors found in this memo is by the Substitute Text command. This command asks you for the correct text and then the text you want replaced (or substituted for). You may specify only one change or several without repeating the command. Statement 3 contains two misspellings:

If you type, you should see:

```
sts3<CR>        BASE C: Substitute C: Text (in) C:
                 Statement (at) A:3
sive<CR>        <New Text> T: sive
cive<CR>        <Old Text> T: cive
n               (Finished?) Y/N:
eme<CR>         <New Text> T: eme
ime<CR>         <Old Text> T: ime
y                (Finished?) Y/N:
                Substitutions Made : 2
                BASE C:
```

Use this command cautiously. You must eliminate ambiguities and avoid causing the system to make substitutions that you don't want. For example in the first substitution if you had specified "e" for "i" instead of "eme" for "ime", the system would have changed ALL occurrences of the the letter "i". Make the text string unique to avoid surprises.

16. To check statement 3 strike \:

The response should look like:

```
BASE C:\
3 A recursive redefinition plan should emerge.
BASE C:
```

17. The memo is finished and you want to make a fresh copy of your file that includes all your changes.

If you type,	you should see:
uf<CR>	BASE C:Update C: File OK:/C: < DIRECTORYNAME, MEMO,NLS;2 > BASE C:

18. A very abbreviated Sendmail session is shown here to enable you to send MEMO to a specific distribution list. NLS has a very extensive system for sending, distributing, cataloging, indexing, and storing documents (files). However, most of these steps are done automatically (and invisibly) for you through the Sendmail system. You begin by going to a subsystem called Sendmail. You give your memo the title Elephant Meeting and then indicate to whom you want it distributed and for what reasons. Then you return to the subsystem Base.

If you type,	you should see:
gs<CR>	BASE C: Goto (subsystem) C: Sendmail OK:
f<CR>	SEND C: File A:
tElephant Meeting <CR>	SEND C: Title T: Elephant Meeting
dajhb<CR>	SEND C: Distribute(for)C: Action(to) T: jhb
didvn<CR>	SEND C: Distribute (for) C: Information (only)(to) T: dvn
s<CR>	SEND C:Send (the mail) OK: Completed
q<CR>	SEND C:Quit OK/C: BASE C:

To name the recipients, you type in their IDENT which is a string of characters that identifies a person. (You typed in your IDENT in Step 1.). This recipient list may be any length; IDENTs must be separated by spaces or commas.

19. The file you just created in NLS has been submitted to the Journal, and a copy has been made for cataloging and future reference purposes. It is not necessary (although permissable) for you to maintain your duplicate version of the file. To delete the file,

if you type,	you should see:
dfmemo<CR> <CR>	BASE C: Delete C: File T:memo OK: Deleted Files are: < DIRECTORYNAME,MEMO.NLS;1 > BASE C:

20. Your work session is over and you leave the system:

If you type,	you should see:
<>1<CR>	BASE C: Logout OK: TERMINATED JOB #,USER DIRECTORYNAME, ACCT ###, TTY # AT DATE TIME USED # in #

HERE IS SOME TNLS COMMAND VOCABULARY YOU HAVE USED AND SOME EASY EXTENSIONS TO IT. THE EXTENSIONS ALL BEGIN WITH THE WORD "TRY" AND INCLUDE SOME EXPLANATION OF THE COMMAND.

More about Help

Typing <CTRL-Q> will give you information based on what you were doing before you typed <CTRL-Q>. Then it will prompt you "T/_:". For more information, type in any term you see or the number of one of the "menu" of subjects that appears below each explanation and then type a <CR>.
If you type _ you will be able to return to the last explanation you were reading. If you say yes by typing "y", you will see this last explanation again. If you say no by typing "n", you will be given the chance to see the previous explanation and so on.

File Manipulation Commands

Create File - creates a new file

Update File - makes a fresh copy of the file with recent changes

Load File - calls up a previously saved file

A Few Useful Control Characters:

<CTRL-X> aborts commands before you have typed <CR>.

<CTRL-Q> gives you explanations about what you were doing and allows you to ask for the meanings of other terms.

<CTRL-E> allows you to continue to insert statements.

Try also:

<CTRL-S> prints out a succinct description of your command.

<CTRL-O> Stops printing.

Creating Text

Insert Statement

Insert Text

Try Insert Word - the text you type is inserted after the point you specify and the system arranges spacing around it for a word.

Editing

Delete statement

Try Delete Text - it requires that you specify the beginning and ending locations of the text you want deleted.

Try Delete Word - you only have to specify one location anywhere in the word you want deleted and spaces, periods, commas, etc. are handled appropriately.

Moving Around In The File

Jump to A: ADDRESS<CR> - moves you to the address specified by ADDRESS.

The ways you have learned to address are:

whole statements by number's;

within statements by "+e" for end of statement, and by content "text", which searches for text in the remainder of the file and if found moves you to the last character of the text you specify.

Seeing Your File

\ - prints the current statement

Try <LF> to print the next statement (<LF> is the Line Feed or LF key on your terminal).

Print Rest - prints from your current statement to the end of the file.

Try Print Statement - it is similar to the "\" command used in Step 6 except that it allows you to specify the address of the (single) statement to be printed and (optionally) certain view control codes such as the one you used in Step 5 to see statement numbers's.

Sending Your File To Other persons

Goto SubsYstem Sendmail

File - sends this file.

Title - gives your item a title

Send for Action--specifies the recipient(s) and that you expect some action.

Send for Information--specifies recipient(s) for information purposes.

Entering/Leaving NLS and TENEX

Login - calls up the TENEX system

NLS - calls up NLS from Tenex

Goto Subsystem - To go from one subsystem to another in NLS

Logout - To leave NLS and TENEX

&SRI-ARC 16-OCT-74 17:23 23911
SRI-ARC 16 OCT 74 23911

TNLS-8 PRIMER

SRI-ARC

16 OCT 74
Augmentation Research Center

STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA 94025

&SRI-ARC 16-OCT-74 17:23 23911
SRI-ARC 16 OCT 74 23911

(J23911) 16-OCT-74 17:23;;;; Title: Author(s): Augmentation Research Center /&SRI-ARC; Distribution: /JOAN([ACTION] please make this part of the DIRT notebook) DIRT([INFO-ONLY]) KWAC([INFO-ONLY] Updated since the architects meeting) ; Sub-Collections: DIRT SRI-ARC NIC KWAC; Clerk: DVN; Origin: < WEINBERG, PRIMER,NLS;14, >, 16-OCT-74 10:43 POOH ;;;;

####;

THE NLS-8 EQUIVALENTS OF THE NLS-7 (Old NLS) COMMANDS 1

To assist those who have been using NLS-7 in changing over to NLS-8 we have prepared this simple alphabetic list of NLS-7 commands with the equivalent NLS-8 command next to them on the right. Both TNLS and DNLS are included. Where a command exists only in TNLS or only in DNLS, it is noted in angle brackets. All NLS-8 commands are in the Base Subsystem unless noted in square brackets. The journal item "New and Changed Features in TNLS-8" (hjournal,31039,) gives a prose account of most of the differencees between NLS-7 and 8. 1a

Where an NLS-8 command is phrased quite differently from the NLS-7 command, it appears in formal NLS syntax. The definitions of the three most important unfamiliar terms in NLS command syntax follow. For more information about formal NLS syntax, see the NLS Command Summary (userguides,summary,) and the Help command. 1b

Definitions: 1b1

SOURCE:
Where NLS syntax requires a SOURCE it usually expects the address of text already online, but you can also type in new text. In TNLS, SOURCE wants either an ADDRESS or an optional TYPEIN of text (prompted by A/[T]:). In DNLS you can also BUG (prompted by T/B/[A]:). When pointing (with BUG or ADDRESS) to Group or Text, two BUGS or two ADDRESSES are needed. 1b2

DESTINATION:
DESTINATION wants you to point to some location in a file. In TNLS, DESTINATION equals ADDRESS (prompted by A:). In DNLS you can also BUG (prompted by B/A:). DESTINATION is used in commands to direct the verb and nominal commandword operators "where" to operate. 1b3

CONTENT:
CONTENT wants you to type in characters, an address, idents, or fileaddress, etc.. You may also put in the address of the content if you precede the address with <CTRL-u>. 1b4

Angle brackets also inclose a few explanatory comments. 1c

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
append	Append Statement
break	Break Statement
copy character	Copy Character
copy word	Copy Word
copy number	Copy Number
copy visible	Copy Visible
copy invisible	Copy Invisible
copy link	Copy Link
copy text	Copy Text
copy statement	Copy Statement
copy branch	Copy Branch
copy plex	Copy Plex
copy group	Copy Group
delete character	Delete Character
delete word	Delete Word
delete number	Delete Number
delete visible	Delete Visible
delete invisible	Delete Invisible
delete link	Delete Link
delete text	Delete Text
delete statement	Delete Statement
delete branch	Delete Branch
delete plex	Delete Plex

NLS-8 Equivalents of NLS-7 Commands

---------- NLS-7 ---------- NLS-8 ----------

delete group ---------- Delete Group

execute assimilate statement - Copy Statement (from) SOURCE (to follow)
DESTINATION OPTION (Filtered:) VIEWSPECS
LEVEL-ADJUST OK

execute assimilate branch --- Copy Branch (from) SOURCE (to follow)
DESTINATION OPTION (Filtered:) VIEWSPECS
LEVEL-ADJUST OK

execute assimilate plex ----- Copy plex (from) SOURCE (to follow)
DESTINATION OPTION (Filtered:) VIEWSPECS
LEVEL-ADJUST OK

execute assimilate group ---- Copy Group (from) SOURCE (to follow)
DESTINATION OPTION (Filtered:) VIEWSPECS
LEVEL-ADJUST OK

execute browse mode enter ---- Set Temporary (modifications)

execute browse mode leave ---- Reset Temporary (modifications)

execute catalog numbers ------ [Sendmail] Reserve

execute connect to terminal <DNLS> ---
Connect (to) Display <DNLS>

execute device type ---------- Simulate (terminal type)

execute edit <TNLS> ---------- Edit Statement <TNLS>

execute file verify ---------- Verify File

execute identification submode <TNLS>:
execute identification status <TNLS> ---
[Sendmail] Show Record (for Ident)
<This is the only function of the old ident
system available to general users in NLS-8,>

execute insert sequential ---- Copy Sequential

execute journal ------------- Goto Sendmail

distribute document<TNLS> - [Sendmail] Forward

hardcopy distribution ----- [Sendmail] Offline

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
insert command form	Insert Sendmail (form)
reenter	Quit To Sendmail <works only in same NLS session; journal work is not saved after Logout> submit [Sendmail] Statement [Sendmail] Branch [Sendmail] Plex [Sendmail] Group [Sendmail] Message [Sendmail] File [Sendmail] Offline
authors	[Sendmail] Authors
clerk	<NO EQUIVALENT--Clerk is automatically assumed to be logged-in user>
Comments	[Sendmail] Comments
distribution	[Sendmail] Distribute
go	[Sendmail] Send
interrogate	[Sendmail] Interrogate
keywords	[Sendmail] Keywords
number	[Sendmail] Number
obsoletes document(s)	[Sendmail] Obsoletes
place link	[Sendmail] Insert Link
process command form	[Sendmail] Process (sendmail form)
status	[Sendmail] Show Status
subcollection(s)	[Sendmail] Subcollections
title	[Sendmail] Title
updates document(s)	[Sendmail] Update

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
execute logout	Logout
execute marker fix	Mark Character
execute marker list	Show Marker (list)
execute marker release	Delete Marker
execute marker release all	Delete All (markers)
execute name delimiter display	Show Name (delimiters)
execute name delimiter statement	Set Name (delimiters in) Statement <AND> Reset Name (delimiters in) Statement
execute name delimiter branch	Set Name (delimiters in) Branch <AND> Reset Name (delimiters in) Branch
execute name delimiter plex	Set Name (delimiters in) Plex <AND> Reset Name (delimiters in) Plex
execute name delimiter group	Set Name (delimiters in) Group <AND> Reset Name (delimiters in) Group
execute ownership of file	Set Link (default for file) <AND> Reset Link (default for file)
execute quit	Quit Nls
execute receive connection <DNLS>	Accept Connect <DNLS>
execute secondary distribution	[Sendmail] Forward

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
execute set control characters <TNLS> ---	[Useroptions] Control (characters) <AND> [Useroptions] Reset Control (characters)
execute show control (mark) <TNLS> ---	<NO EQUIVALENT>
execute show selections <TNLS> ---	<NO EQUIVALENT>
execute show upper case <TNLS> ---	<NO EQUIVALENT>
execute status control characters ----	[Useroptions] Show Control (characters)
execute status file ----------	Show File Status
execute status link stack ----	Show File Return (ring)
execute status viewspecs <DNLS> ---	Show Viewspecs (status)
execute tabstops set <DNLS>---	[Useroptions] Printoptions Tab <AND> [Useroptions] Reset Printoptions Tab
execute unlock file ----------	Delete Modifications
execute viewchange <TNLS> ----	[Useroptions] Currentcontext (length) <AND> [useroptions] Reset Currentcontext (length)
execute viewchange printing (parameters) ---	[Useroptions] Printoptions <AND> [Useroptions] Reset Printoptions
execute viewchange control (characters) ---	[Useroptions] Reset Control (characters)

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
execute viewchange feedback <TNLS> ---	[Useroptions] Feedback <AND> [Useroptions] Reset Feedback
freeze statement <DNLS> ------	Freeze Statement <DNLS>
freeze statement release <DNLS> ---	Release Frozen (statement) <DNLS>
freeze statement (release) all <DNLS> ---	Release All <DNLS>
goto control playback <DNLS> -	Playback Record (of session)
goto control quit <DNLS>------	Stop Record (of session)
goto control record <DNLS>----	Start Record (of Session)
goto display clear (display area) <DNLS> ---	<NO EQUIVALENT>
goto display format (display area) character (size) <DNLS> ---	Set Character (size)
goto display horizontal (split) <DNLS> ---	Insert Edge
goto display move (boundary) <DNLS> -	Move Edge <AND> Delete Edge
goto display tty-simulation (window)<DNLS>---	Set Tty (simulation window) <AND> Reset Tty (simulation window)
goto display vertical (split) <DNLS> ---	Insert Edge
goto exec -------------------	Goto (subsystem) Tenex
goto merge branch -----------	Merge Branch

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
goto merge plex	Merge Plex
goto merge group	Merge Group
goto NIC resource query <TNLS>	<NO EQUIVALENT>
goto programs buffer (size)	[Programs] Set Buffer (size) <AND> [Programs] Reset Buffer (size)
goto programs content (analyzer)	Set Content To <AND> [Programs] Compile Content (pattern)
goto programs deinstitute	[Programs] Deinstitute
goto programs execute (program)	[Programs] Run Program
goto programs institute	[Programs] Institute program
goto programs L10 (user program compile)	[Programs] Compile L10
goto programs pop	[Programs] Delete Last (program)
goto programs reset	[Programs] Delete All (programs)
goto programs status	[Programs] Show Status
goto query <TNLS>	<NO EQUIVALENT>
goto sort group	Sort Group
goto sort plex	Sort Plex
goto sort branch	Sort Branch
goto use (measurements)	<NO EQUIVALENT>
insert character	Insert Character
insert date	Insert Date

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
insert date time	Insert Time (and date)
insert word	Insert Word
insert number	Insert Number
insert visible	Insert Visible
insert invisible	Insert Invisible
insert link	Insert Link
insert text	Insert Text
insert statement	Insert Statement
jump (to) ahead <DNLS>	<NO EQUIVALENT>
jump (to) down <DNLS>	Jump (to) Down
jump (to) end (of item) <DNLS>	Jump (to) End (of Branch)
jump (to) head <DNLS>	Jump (to) Head
jump (to) item <DNLS>	Jump (to) Item <AND> <DNLS> Jump (to) Address (relative to)
jump (to) file ahead <DNLS>	<NO EQUIVALENT>
jump (to) file return <DNLS>	Jump (to) File Return
jump (to) link <DNLS>	Jump (to) Link
jump (to) origin <DNLS>	Jump (to) Origin
jump (to) predecessor <DNLS>	Jump (to) Predecessor
jump (to) return <DNLS>	Jump (to) Return
jump (to) successor <DNLS>	Jump (to) Successor
jump (to) tail <DNLS>	Jump (to) Tail
jump (to) up <DNLS>	Jump (to) Up

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
load file	Load File
move character	Move Character
move word	Move Word
move number	Move Number
move visible	Move Visible
move invisible	Move Invisible
move link	Move Link
move text	Move Text
move statement	Move Statement
move branch	Move Branch
move plex	Move Plex
move group	Move Group
null (file)	Create File
output assembler (file)	Output Assembler
output compiler	[Programs] Compile File
output device COM	Output Com
output device printer (file)	Output Printer
output device sequential (file)	Output Sequential File
output device teletype <TNLS>	Output Terminal
output device XCOM	Output Com Test
output file	Update File Compact
output quickprint (file)	Output Quickprint

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
output sequential	Output Sequential File
print CA <TNLS>	<TNLS> Print Rest
print journal <TNLS>	<TNLS> Print Journal (mail)
print branch <TNLS>	<TNLS> Print Branch
print statement <TNLS>	<TNLS> Print Statement
print plex <TNLS>	<TNLS> Print Plex
print group <TNLS>	<TNLS> Print Group
quit	Quit Nls
replace character	Replace Character
replace word	Replace Word
replace number	Replace Number
replace visible	Replace Visible
replace invisible	Replace Invisible
replace link	Replace Link
replace text	Replace Text
replace statement	Replace Statement
replace branch	Replace Branch
replace plex	Replace Plex
replace group	Replace Group
substitute	Substitute
transpose character	Transpose Character
transpose word	Transpose Word
transpose number	Transpose Number

NLS-8 Equivalents of NLS-7 Commands

NLS-7	NLS-8
transpose visible	Transpose Visible
transpose invisible	Transpose Invisible
transpose link	Transpose Link
transpose text	Transpose Text
transpose statement	Transpose Statement
transpose branch	Transpose Branch
transpose plex	Transpose Plex
transpose group	Transpose Group
update (file)	Update File
view (set) <DNLS>	Set Viewspecs
viewspecs change <TNLS>	Set Viewspecs
viewspecs reset <TNLS>	Reset Viewspecs <To default controlled by Useroptions>
viewspecs status <TNLS>	Show Viewspecs
xset character	Force (Case) Character
xset word	Force (Case) Word
xset visible	Force (Case) Visible
xset invisible	Force (Case) Invisible
xset link	Force (Case) Link
xset text	Force (Case) Text
xset statement	Force (Case) Statement
xset branch	Force (Case) Branch
xset plex	Force (Case) Plex
xset group	Force (Case) Group

NLS-8 Equivalents of NLS-7 Commands

---------- NLS-7 ---------- NLS-8 ----------

xset mode lower ---------- Force (Case) Mode Lower

xset mode capital ---------- Force (Case) Mode Upper

xset mode initial ---------- Force (Case) Mode First

SP ADDRESS <TNLS> ---------- <TNLS> Jump (to) Address

LF <print next statement> <TNLS> ---
<TNLS> LF

; <comment> <TNLS> ---------- ; <comment>

. <print location of CM> <TNLS> ---
<TNLS> .

\ <print statement> <TNLS> -- <TNLS> \

/ <print context of CM> <TNLS> ---
<TNLS> /

" <print back statement> <TNLS> ---
<TNLS> "

TNLS-8 EQUIVALENTS OF NLS-7 COMMANDS

SRI-ARC

16 OCT 74

Augmentation Research Center

STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA 94025

NLS-8 Equivalents of NLS-7 Commands

(J23913) 16-OCT-74 17:13;;;; Title: Author(s): Augmentation Research Center /&SRI-ARC; Distribution: /JOAN([ACTION] Please make this part of the DIRT notebook) DIRT([INFO-ONLY]) SRL([INFO-ONLY]) JMB([INFO-ONLY]) JHB([INFO-ONLY]) KWAC([INFO-ONLY]) ; Sub-Collections: SRI-ARC DIRT NIC KWAC; Clerk: DVN; Origin: < VANNOUHUYS, SPLTLVOLDNW,NLS;1, >, 16-OCT-74 17:03 POOH ;;;; Title:

####;

RLL 3-SEP-74 15:15 23916

bug ussng jump to name external wih split screens.

(J23916) 3-SEP-74 15:15;;;; Title: Author(s): Robert N. Lieberman/RLL; Distribution: /FDBK([ACTION]) ; Sub-Collections: SRI-ARC; Clerk: RLL;

RLL 3-SEP-74 15:15 23916

bug ussng jump to name external wih split screens.

FST entry nonextistent message with pushdown overflow t 34013. This happened when I tried a jump to name external command for a link in the file of links. The situation was during a three window configuration: one vertical split and the left side split horizontally, the jump was to load the lower left window. I tried this three (3) times with exactly the same results. (ussng NLS not work of course). Rob

DVN 4-SEP-74 08:42 23917

No way to Output SID's

(J23917) 4-SEP-74 08:42;;;; Title: Author(s): Dirk H. Van Nouhuys/DVN; Distribution: /SRL([INFO-ONLY]) JCN([INFO-ONLY]) EKM([INFO-ONLY]) NDM([INFO-ONLY]) ; Sub-Collections: DPCS SRI-ARC; Clerk: DVN;

DVN 4-SEP-74 08:42 23917

No way to Output SID's

I don't know of any way to turn on SID's with directives. Of course there should be one. Quickprint will reproduce just what you see on the sccreen. There is a rather complecated way of having serial numbers (numbers the output procesor creates by counting) attached to statements. You can see it set up in the the header of (hardy, workstations,) and come back to me for more explanation if you want to use it. 1

Agenda for Second KWAC Meeting, September 9-13, 1974

(J23919) 4-SEp-74 19:12;;;; Title: Author(s): James C. Norton/JCN;
Distribution: /KWAC([ACTION]) RSR([ACTION]) CHI([ACTION])
RWW([ACTION]) ; Sub-Collections: SRI-ARC KWAC; Clerk: JCN;
Origin: < NORTON, KWACAGENDA.NLS;3, >, 4-SEP-74 19:02 JCN ;;;;
####;

Agenda for Second KWAC Meeting, September 9-13, 1974

Monday 9/9		1
9:00	Welcome and Agenda Discussion -- Norton	1a
10:00	Introduction of Architects and ARC Applications Staff,	1b
11:00	The Architect Community (KWAC) and Roles -- Engelbart	1c
12:30	Lunch at the I-Building	1d
2:30	Remote Meeting, Recreation, Repasts at Jim Norton's House (Architects and their ARC Workshop Utility Contacts)	1e
Tuesday 9/10		2
9:00	Talks by Architects: Experiences to date (Stone, Mattiuz, McLindon,...)	2a
11:00	Open Discussion	2b
12:00	Lunch - with ARC Staff	2c
1:30	NLS-8 Introduction: What is Changed/Added and Why -- Irby	2d
2:30	NLS-8: Training, Discussion, Use (by experience groups)	2e
Wednesday 9/11		3
9:00	ARC Development, NSW project: Status and Plans -- Watson	3a
10:30	ARC Applications: Status and Plans -- Norton	3b
12:00	Lunch - with ARC Staff	3c
2:00	More New NLS-8: Training, Discussion, Use	3d
Thursday 9/12		4
9:00	Discussion of Other Communities: Management, DDPCS, ...	4a
10:30	Discussion of KWAC Local Training Plans	4b
12:30	Lunch	4c
2:00	Discussion of Plans for KWAC: Activity and Next meeting	4d
4:00	Special Social Evening for Architects at Doug Engelbart's House	4e
Friday 9/13		5
9:00	Discussions of Local Applications and plans	5a
12:30	Lunch	5b
2:30	Concluding Discussions	5c

Agenda for Second KWAC Meeting, September 9-13, 1974

WORKSHOP ARCHITECTS:			6
ARPA	Connie McLindon	CKM	6a
BRL	Stan Taylor	SMT	6b
Bell	Inez Mattiuz	IMM	6c
ETS	Brian McNally	BJM	6d
Hudson	Rudy Ruggles	RLR	6e
MIT-Seismic	Bob Sheppard	RMS2	6f
NIC-OPER	Jake Feinler	JAKE	6g
NSRDC	Frank Brignoli	FGB	6h
RADC	Duane Stone	DLS	6i
SRI	Mike Placko	MAP2	6j
ARC APPLICATIONS WORKSHOP UTILITY STAFF:			7
ARC Director	Doug Engelbart	DCE	7a
Manager/Liaison	Jim Norton	JCN	7b
Contracts/Hardware	Martin Hardy	MEH	7c
User Development	Jim Bair	JHB	7d
	Susan Lee	SRL	7e
Software/User Help	Dave Hopper	JDH	7f
	Dean Meyer	NDM	7g
Marketing	Robert Lieberman	RLL	7h
TENEX/User Help	Jeff peters	JCP	7i
Consultant	Bob Ratner	RSR	7j

IBM Request for Line Processor Papers

(J23920) 5-SEP-74 10:19;;;; Title: Author(s): Elizabeth K. Michael/EKM; Distribution: /EKM([ACTION]) BC([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: EKM; Origin: < MICHAEL, LET,NLS;3, >, 5-SEP-74 09:46 EKM ;
;;;####;

EKM 5-SEP-74 10:19 23920

IBM Request for Line Processor Papers

Augmentation Research Center

Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, California 94025

Thomas M. Hadley
IBM Data Processing Division
444 East College Avenue
State College Pennsylvania
Pennsylvania 16801

Dear Mr. Hadley:

I have enclosed preprints of the two papers on the ARC line processor, the device we use to interface the mouse and keyset to standard display terminals. 1

C. H. Irby, "Display Techniques for Interactive Text Manipulation", 1a

D. I. Andrews, "Line Processor: A Device for Amplification of Display Terminal Capabilities for Text Manipulation" 1b

Both papers have been published in the AFIPS Conference Proceedings, Volumn 43, 1974, National Computer Conference, 2

We would be happy to provide any additional information you might want about the line processor or our NLS system in general. 3

Sincerely,

Elizabeth K. Michael
Augmentation Research Center

KIRK 5-SEP-74 17:26 23921

NP for Move Edge

(J23921) 5-SEP-74 17:26;;;; Title: Author(s): Kirk E. Kelley/KIRK; Distribution: /NP([ACTION]) ; Sub-Collections: SRI-ARC NP; Clerk: KIRK;

KIRK 5-SEP-74 17:26 23921

NP for Move Edge

Moving Edge to the margin should not delete the window, but should save it for later use. 1

KIRK 5-SEP-74 20:40 23922

NP for jump file return

(J23922) 5-SEP-74 20:40;;;; Title: Author(s): Kirk E. Kelley/KIRK;
Distribution: /FDBK([ACTION]) ; Sub-Collections: SRI-ARC; Clerk:
KIRK;

KIRK 5-SEP-74 20:40 23922

NP for jump file return

Jump to file return choice message (currently a filelink) should be the complete link containing the statement number and viewspecs. 1