JGM 23-JUN-71 11:33 7340

The Modular Programming System: Processes and Ports

this is the first issue of basic notions and implementation notes of he MPS project

Processes and Ports: 2

Basic Notions 2a

An atomic process is an executable instance of a program and an environment (private data, state information, a stack and "connections" to other processes). Separate processes can communicate control or information or both among themselves. The primary means of inter-process communication are called "entry-ports", (non-entry or normal) ports -- hereafter, "port" means non-entry port. Both control and communication can be transferred over ports. 2a1

Creating a Process 2b

An atomic process can be created by loading a "module", which module contains machine code and an initial environment for the process. A name is also given to the process to distinguish it from other instances of the same module. Internally, a process consists of three distinct segments [see the document (deutsch,docseg,:wn) for a description of the software segmentation machinery for the Modular Programming System (MPS)]. There is a code segment, which is shared by all the incarnations of that module; a data segment, one for each instance of the module (i.e., one per process) which contains the static storage for the process; and a stack segment which acts as the local variable and procedure call stack for the process. The phrase "data segment of a process" and "process" are used interchangeably since there is an isomorphism between them. 2b1

All the programs running in such a system are (at least conceptually) processes. When one process causes another to be created, it is designated as the "owner" of that new process. 2b2

If something happens to a process which it is not prepared to handle, control will be given to that process's owner so that it can attempt to take care of the problem. Any process is free to create another: hence, there is a "tree" of owners at any moment in the system. The root of that tree is a process having no owner which we will call SYSTEM. 2b3

In order to allow a group of processes to cooperate in performing some function they must 2b4

(a) be created 2b4a

(b) be connected so that control and information may be passed among them, and 2b4b

(c) be "started", one at a time to begin the task which that "configuration" of processes is to perform. 2b4c

The CREATE Statement: 2c

A process can cause a module to be instantiated as a process by the CREATE statement: 2c1

```
[procvar '←] "CREATE" modulename ["AS" processname];
```
2c1a

This causes incarnations of the named module's code, data and stack segments to be created. The code segment is shared with any other instances of this module. The process's data and stack segments are created and initialized. If no process name is provided, the system will generate one (probably some mystical but unique number) as the name of the data segment. The stack segment name will, at least initially, always have an internally generated, unique name. 2c2

The process requesting a CREATE has a predeclared, standard port called its OWNER port, connected to the created process's START port (also standard in every process) as a result of the CREATE - see the discussion on starting a process [STARTUP] below for more detail on this. Also, if the "procvar←" phrase is present, a reference to the created process (i.e., to its data segment) will be stored in procvar. 2c3

Each process possesses, in addition to its OWNER and START ports, a normal port called its FAULT port which is used to communicate problems encountered in the process to a process called its "owner" which is responsible for it. That is, the FAULT port's connection defines who is the owner of a process. The initial owner is its creator, and the FAULT port in a newly created process is connected to the OWNER port of its creator as a side effect of the CREATE statement. 2c4

JOINing Processes 2d

For purposes of explication we will denote a port "a" which belongs to some process p as p:a. Port names are considered local to the process in which they are declared. Thus p and q, both processes, may possess ports a and b respectively by which they are to cooperate: i.e., p:a is to be joined with q:b. But it is intended that p and q view their respective ports as virtual facilities whose connection to some real facility will be decided by a third process (normally the owner of one of the processes). 2d1

The means for connecting p:a to q:b is the JOIN statement: 2d2

JOIN p:a TO q:b 2d2a

This particular statement only specifies that information of the wherabouts of q:b is stored in p:a and not the opposite. If q:b is to "know" about p:a then it is necessary to also say
JOIN q:b TO p:a 2d2a1

For convenience, "JOIN p:a AND q:b" is used to denote that p:a is to be connected to q:b and vice versa. 2d2a1a

A port and its connection information is called a "path" from the subject process to the object process in which the object port resides. 2d3

Running Processes 2e

Processes run in a completely synchronous manner with exactly one process running at any given moment. Normally a process temporarily suspends execution by sending information and control over a port to the process whose port is attached to the other end. For convenience in describing this and similar situations we will call the process which is running and in the act of passing control the "subject" process (and its ports subject ports) and the process connected to the other end of the subject process's port (to whom control will be passed) the "object" process (his ports are called object ports). When a process sends control and (possibly) information across a port it is said to make a "port call" on that virtual facility. 2e1

(STARTUP) Starting a Process 2f

A process which has never run is in the "stopped" state. A stopped process may only become "active" by receiving control over one of its entry ports. Each process possesses a standard entry-port called START, and may possess other entry-ports if declared at compile time. 2f1

The information associated with an entry-port is 2f2

an address within the process where execution is to begin whenever control arrives over the entry port, called its "entry-point", 2f2a

a message buffer where any message to the entry port is to be placed, 2f2b

the address of the object to which the entry-port is connected (it may be unconnected or connected to either an entry-port or a normal port in some process) 2f2c

An entry-port is declared by a statement in the program of the form 2f3

```
portname:    ENTRY PORT [ '( messageid ') ];
```
2f3a

and the special entry-port START need only be declared if the program wants to accept a message on the START port. If START is not explicitly declared, it is as if the following statement were inserted before the first executable program statement: 2f4

```
ENTRY PORT START;
```
2f4a

Basically, when control reaches a stopped process over an entry-port, the process's status is changed to "active" and its program counter (PC) is set to the entry-point value of the entry-port. The process will revert to the stopped state when a "STOP message" statement is executed or the process attempts to use an entry-port of its own. Indeed, "STOP" is equivalent to using the entry-port over which control arrived most recently. 2f5

Whenever a process attempts to use an unconnected port (entry or non-entry), control is sent to that process's "owner". 2f6

The owner of a process is defined by the connection of that process's FAULT port. Whenever a process generates a fault which it is not prepared to handle, a port call on its FAULT port is simulated by the system. A message which indicates the cause of the fault is sent over the port to the owner process. All the normal control mechanisms for port calls are true for the simulated call on the FAULT port. Naturally, any attempt to disconnect a process's FAULT port will cause an error to be generated in the running process. 2f6a

Assume process c is the owner of process p. Then c can cause p to become active by a statement of the form 2f7

```
"RUN" process [ ': portname ] [ '( message ') ];
```
2f7a

E.g., RUN p; 2f7a1

If portname is omitted, the process's START entry-port is assumed. However, the portname may specify either an entry-port or a normal port in the object process. Only the entry-port case will be discussed at this point. 2f7b note that if the RUN statement is executed after one create and before any other CREATE's are done, then it is equivalent to the owner process issuing the following port call: 2f7b1

PORT OWNER ['(message')]; 2f7b1a

Assume that c executes the statement
RUN p:e
where e is an entry-port of p, and p is stopped. Then, c is suspended and p is made active with execution commencing at e's entry point. 2f8

If RUN p:e is executed but p is not in the stopped state, the following occurs: 2f9

before p is made active, its RECENT'EP word is copied into SAVED'RECENT (see PORTCONTROL) and the connection information in the entry port is copied into SAVED'CNCTN. 2f9a

p's base registers are loaded, and p begins execution at the point specified by the entry-point value associated with the entry port. The previous saved value of PC is undisturbed. 2f9b

Saving RECENT'EP and the connection information for the entry port over which control arrived is done to allow recursive use of a process. Copies of these specific cells are made by the system because they are the only ones which are overwritten in the process of entry port entry. All other information in the process's data segment can be pushed down by the process itself once it regains control using the statement: 2f10

"PUSH" "ENVIRONMENT"; 2f10a

This statement makes a copy of the process's current environment (i.e., its data segment) onto its stack: this includes the stored PC-value and base registers. The data segment is then linked to this copy via a fixed cell (OLD'ENV) in the data segment and the stack base value in the process is updated to point past the end of the data segment copy in the stack segment. If the PUSH ENVIRONMENT statement is done before any port calls, the PC-value saved with the copied environment is the one which would have been used had control arrived over a normal port. The new environment is then a copy of the previous (in fact, it is the previous environment -- the chunk on the stack is the copy) and all of the process's neighbour processes are always connected to its current environment. 2f11

Later p may execute a "POP ENVIRONMENT" statement - which essentially reverses a PUSH ENVIRONMENT - and then leave via an entry-port. Making a port call on an entry port does the following: 2f12

RECENT'EP$CONNECTION ← SAVED'CONNECTION; 2f12a

RECENT'EP ← SAVED'RECENT; 2f12b

the actions of a normal port call (see the sequence NN1-NN5 below) 2f12c

This assures that the process reverts to the state which existed prior to control arrival over an entry-port. 2f13

Using Normal Ports 2g

Messages in Ports 2g1

Each port in a process possesses a message buffer which may contain either the null message (nullmsg) or some valid message. The buffer's contents can be moved to a variable, or simply destroyed by the following statement: 2g1a

[variable '←] "EMPTY" portname ; 2g1a1

If the optional phrase is not present, the message buffer for the iort is set to contain the null message. If the message buffer for a port is empty (i.e., contains the null message) and the process attempts to empty that port, an error results. This error can be handled by appending an "error phrase" to the EMPTY statement (see error'phrases). 2g1b

Port Calls: 2g2

Normally, a message is only put into a port when control is passed from the sender to the receiver over that path. A process can send control and (optionally) a message over a port using a statement of the form: 2g2a

[lhs '←] "PORT" portname ['(message')]; 2g2a1

Executing such a statement will cause the following sequence of actions: 2g2b

(NN1) the "state" of the subject process is saved in its static environment or data segment; the portion of the state which is saved includes the value of the PC, and the stack pointer and local variable or frame pointer if the process has them. 2g2b1

(NN2) The object port is made to point to the subject port; this is called railroad switching and is explained below. 2g2b2

(NN3) The given message, if present, is placed in the object process's message buffer; if no message is present, the null message is placed in the object port's message buffer. 2g2b3

(NN4) The address of the object process's data segment is loaded into a base register from the object port. 2g2b4

(NN5) The object process's stack and frame pointers, the base address of its code segment, and any other required base registers are loaded from its data segment, and the PC value is used to start the process in execution: 2g2b5

(a) The PC may be valid and point somewhere in the code segment for the object process: in this case the process simply resumes execution. 2g2b5a

(b)The PC may be the address of a system routine which initiates the signalling of "control faults": a process which is in state "stopped" has this address as its PC value. For a complete description of the result of signalling a control fault see the section SIGNALS. 2g2b5b

(NN6) When control comes back to the subject process (by the execution of this same sequence of actions on the object process side), the message buffer contents may be stored in the "lhs" variable, if present. If it is present but the port's message buffer contains the null message, a "nomessage" signal will be generated. See the description of the EMPTY statement below for more detail of this. 2g2b6

Control normally returns to a process over the same path by which it left. It may, however, return over a different path; the process may determine over which path control returned by executing the system function RECENT'PORT( ) which returns the address of the port concerned as its result. 2g2c

The object port is set to point at the subject port in setp NN2 so that control can later return over that path from the object process. This switching is necessary because many ports may connect to a single port and control can only return from that single port to exactly one of the ports connected to it. The one from which it gained control most recently is the obvious choice. 2g2d

It is not necessary to take the message from a port when control arrives over the port. The contents of a port's message buffer can be removed and th@e null message put into the buffer by a statement such as 2g2e

```
  [ lhs '←] "EMPTY" portname;                              2g2e1
```

If the lhs is not present, the null message is simply written into portname's message buffer (specified as portname$Message in MPL(A)). If the lhs is present, this statement is equivalent to 2g2f

```
  IF portname$Message # NullMsg  % %                       2g2f1
    THEN                                                   2g2f1a
      BEGIN                                                2g2f1a1
        lhs ← portname$Message;                            2g2f1a1a
        portname$Message ← NullMsg;                        2g2f1a1b
      END                                                  2g2f1a2
    ELSE SIGNAL NoMessage;  % see section SIGNAL  %        2g2f1b
```

A "CATCH-phrase" may be attached to the EMPTY statement to field any possible generated NoMessage signal (see SIGNALS). 2g2g

If a port, b, is considered bidirectional, it can be used by writing 2g2h

```
  in ← PORT b(out);                                        2g2h1
```

Assuming that a message returns along with control over b after the port call, the assignment operator will simply move the received message into the variable in. This is equivalent to 2g2i

```
  PORT b(out);  in ← EMPTY b;                              2g2i1
```

As mentioned previously, a CATCH-phrase may be appended to a port call statement to handle the case when the null message is unexpectedly received. 2g2j

Control may also enter a process over a normal port from an unconnected parent process by means of the RUN statement. Except for the fact that the connection information in a port, b, is unchanged by RUN p:b, the effect is exactly as if control had returned to p across the port b from the object process to which b is connected. This provides a means of jolting processes to life after port or control faults as well as allowing the creator process to intercede in a created configuration of processes. If a message is supplied with the RUN statement; e.g.,

```
                    RUN p:a (message);
```

the message is put into a's message buffer as if it were being received over the port. 2g2k

If, in a configuration some of the ports on various processes are not needed for a specific application, they may be specified to be "ignored". An ignored port is one which has been JOINed to itself. Thus, when a port call is made on one, the subject process is also the object process and resumes without control ever leaving. Any messages sent over an ignored port, therefore, will appear in its own message buffer (this last is of no special importance: it is simply what will happen). 2g2l

(SIGNALS) Simple Signal Phrases and Actions 2h

A signal can be generated by a SIGNAL statement in a procedure: 2h1

"SIGNAL" [code] ['(paramlist')]; 2h1a

or, by the occurrence of events such as machine traps (e .g., arithmetic overflow). 2h2

Once a signal has been generated, no matter by what means, some action must be taken by some program before normal control can resume. The main problems with signals concern who is eligible to "catch" a signal and what he may do when given control. 2h3

A signal is first propagated back through the procedure call hierarchy in the running process in which the signal was generated. The first procedure encountered in this backwards search which indicates its willingness to catch the signal is given control. 2h3a

A procedure declares itself a candidate signal-catcher by providing a CATCH-phrase (or sequence of CATCH-phrases) which will inspect a generated signal when requested during the backwards scan through the procedure call hierarchy and either accept the signal or reject it. Rejecting it will cause the backwards scan to continue; accepting it allows the CATCH-phrase to take some simple action, after which the normal flow of control will resume in the procedure containing the CATCH-phrase. 2h3b

The syntax of a CATCH-phrase is 2h3b1

catchphrase = "CATCH" [lhs] '($(caserel ': erroraction ';) '); 2h3b1a error actions will be described shortly; caserel means what it normally does in MPL(A), except that the value being compared in each binary relation (caserel) is the signal value. If the optional lhs is present, the value of the signal is assigned to it. 2h3b2

A CATCH-phrase is "provided" as a potential signal catcher either by the execution of an ENABLE statement or by appending the phrase to a statement [and to individual operators in some later version] 2h3c

The ENABLE statement has the syntax: 2h3d

[label ':] "ENABLE" (labelid / catchphrase); 2h3d1

[can an ENABLE statement have a catch phrase attached to it?] 2h3d1a

The CATCH-phrase enabled is either the one appended to the ENABLE clause or the CATCH-phrase in another ENABLE statement identified by the labelid. When an ENABLE statement is executed during normal execution, the address of the CATCH-phrase is pushed onto a (linked) "CATCH-stack" associated with that incarnation of the procedure. If the CATCH-phrase is already enabled (and therefore already has an entry in the catch-stack), it is first removed from its previous position before being pushed onto the top of the stack. The catch-phrase is then a possible signal catcher until control returns from that incarnation of its procedure, or until a CANCEL statement causes it to be removed from the catch-stack (the description of CANCEL follows). 2h3e

A catch-phrase attached to some (non-CATCH) statement is a potential signal catcher only during the execution of the statement: it is automatically ENABLEd at the start of that statement and CANCELed on its successful completion. 2h3f

Simple Catcher Determination and Actions 2i

A catch-phrase can list a set of specific codes, "classes" of codes or "all codes" on which it is prepared to act. The actions which it may take on a given error or class of errors is one of the following: 2i1

(a) an arbitrary statement. 2i1a

(b) VALUE expression: this action takes the value of the expression as the value of the called procedure and execution of the receiver will continue in the same manner as it would on a normal return from the called procedure. 2i1b

In both cases (a) and (b), before the error action is executed, the call stack is cut back to the same point it would have been at on a normal return to the receiver. 2i1b1

(c) "INVOKE" procedure call: in this case, the call stack remains as it was when the error was generated, and the procedure in the error action is called "almost as if" it had been called by the error generator. 2i1c

Signals Between Processes 2j

Signal Messages across Ports 2j1

No SIGNAL facilities are provided for processes talking to one another across ports (with the exception of the OWNER/FAULT paths). However, since errors can occur in attempting to use a port (connection, or control fault) a catch-phrase can be appended to a port call to field such conditions within the running process. Once generated, such a signal looks like any other and could be fielded by any pocedures in the call hierarchy of the running process. 2j1a

The FAULT-OWNER Chain as a Signal Path 2j2

When any signal is not fielded by a process itself, it is propogated up the FAULT/OWNER chain in an attempt to find someone to accept it. In each process, the signal passes through the same stages that any signal would. When it is finally fielded, that process's OWNER port is JOINed to the FAULT port of the process at which the signal originated. 2j2a

This control scheme is closely analogous to the scheme within a process. 2j2b

(PORTCONTROL) Port Control: Code and Semantics 3

Layout of The Data Segment of a Process 3a

```
DSEG:SEG'NUMBERS: XWD        dsegn,csegn                          3a1

LINK'CODE:        XWD        linkbase,codebase                    3a2

LOCPTR:           XWD        0,RETLOC                             3a3

RETLOC:           MOVE        C,LINK'CODE                         3a4

          or      MOVE        C,nonxmem                           3a4a

                  MOVS        LNK,C                               3a5

                  JRSTF         @-2(S)                            3a6

PC:                XWD    0,pc'value                              3a7

RECENT'PORT:          XWD   0,0      ;ptr to most recent
entry-port over which control arrived                             3a8

SAVED'RECENT:     XWD        0,0       ;recent'port saved here    3a9

* the name of this process:                                       3a10

MYNAME:   ASCII   'process name'                                  3a11

          ASCII   'process name'                                  3a12

FAMILY:   XWD     son'list,brother'link                           3a13
```

son'list=0 means that this process has no son processes. If brother'link=0, this is the last process on its parent's son list. Both these pointers refer to the beginning of data segments. 3a13a

```
*          the process's start port (an entry port)               3a14

START:            XWD        START,trap'port                      3a15
```

trap'port is a "port" in the system which is used to field port faults. Any unconnected port is, in reality, connnected to the trap'port. 3a15a

```
                  XWD        0,0          ;message word for the
START port                                                        3a16

                  XWD        0,DSEG                               3a17
```

```
                    JRST        0,EPENTER                          3a18
```

EPENTER is a system routine which handles control arrival over an entry port. 3a18a

```
                    XWD         0,entry'point                      3a19
```

```
*          the process's fault port                                3a20
```

(FAULTPORT) 3a21

```
FAULT:              XWD         FAULT,object'port                  3a22
```

Object'port represents a pointer to the port to which this port is connected. 3a22a

```
                    XWD         0,0        ;port's message buffer  3a23

                    XWD         0,DSEG                             3a24

                    JRST        @PC(D)                             3a25
```

This word distinguishes a normal port from an entry port. The address which it contains is used in the port call mechanism. Cf. (PORTCALL). 3a25a

```
*          storage for the registers                               3a26

REG'BASE:FRAME:     XWD         0,frame'ptr                        3a27

STK'PTR:            XWD         max'stack,stack'ptr                3a28
```

Any other base registers which the process needs to have loaded are placed following STK'PTR 3a28a

The process to which this process's fault port is connected is defined as the owner of this process, and is assumed responsible for him. 3a29

Code for: [var '←] "PORT" port ['(message')]; where "port" is a normal port 3b

In-line code: 3b1

```
                 HRLZI     M,400000    ; the null message          3b1a

        or          MOVE        B,message(D)  ;if the optional
      (message) phrase is present                                  3b1a1

                 MOVE       B,port(D)                              3b1b
```

```
              JSP         P,portcall                                 3b1c

    global code:                                                      3b2

      portcall:   MOVEM        S,STK'PTR(D) ;save stack pointer
      word                                                           3b2a

                  MOVEM        F,FRAME(D)    ;save current frame
      pointer                                                        3b2b

      send'no'stk: MOVEM       P,PC(D)        ;save pc               3b2c

                  MOVSM        B,(B)         ; railroad switching    3b2d

                  MOVE         D,dseg'ptr(B) ;get pointer to
      object port's dseg                                             3b2e

                  MOVE         C,@RETLOC(D)  ;load codebase and
      linkbase and check for not-in-memory trap                      3b2f

                  JRST         STARTUP(B)    ;resume the object
      process                                                        3b2g
```

port layout: (see also examples in dseg layout above, esp. FAULTPORT) 3b3

```
      port:       XWD         port,object'port                       3b3a
```

If the port is not connected, object'port is replaced by a a pointer to a "fake" system port called pf'port which will cause control to enter a port-fault error routine using the normal port call machinery to get there. 3b3a1

The port may also be specified as an ignored port: any uses of it act as null operations. This is handled by joining the port to itself: then any use of the port simply causes the process which is shutting down to be immediately resumed. 3b3a2

```
      msg:        WORD                     ;message word             3b3b

      dseg'ptr:   XWD         0,DSEG                                 3b3c
```

note that this word must be set up for each port in the dseg whenever a copy of the process is created. 3b3c1

```
      startup:    JRST        @PC(D)  ;if process has no base
      registers at all                                               3b3d
```

Alternates, depending on the process and the port, are the following: 3b3d1

Normal port, process with base registers: 3b3d1a

```
ZWD        load'base+i                                  3b3d1a1
```

where load'base is a global routine. If the process only has stack and frame base registers, load'base+2 is used, for instance. 3b3d1a1a

entry port, process with or without base registers: 3b3d1b

```
JRST       epenter+i                                    3b3d1b1
```

this is used when the port is an entry port. It also performs the function of load'base+i. epenter+0 is used when the process has no base registers to be loaded. 3b3d1b1a

```
entry'point:  XWD      0,entry'point'value   ; only
present for an entry port.                                3b3e
```

The support code for port call involves a system routine called load'base above. If the process needs to have registers i through 17 restored before it resumes execution, each normal port will have 3c

```
JRST load'base+i                                          3c1
```

in its startup word. If the process has no base registers other than its stack pointer and frame pointers, it will use load'base+16; 3d

In general, if the process requires i base registers, they must be registers 17,...,17-i+1. These registers are laid out in the process's DSEG in the order F,S,13,...,17-i+1, and only as many words as are necessary need be reserved in the DSEG. Also, since this region is variable, it is the last part of the DSEG which must be present for every process; everything in front of it is fixed in size. 3e

The routine load'base has the following form : 3f

```
load'base+i:   MOVE   i,REG'BASE+17-1(D) ; load register i    3f1

               ...                                             3f2

               ...                                             3f3
```

```
                MOVE     S,STK'PTR(D)  ;load'base+16            3f4

                MOVE     F,FRAME(D)    ; the last base register 3f5

                JRST ,   @PC(D)        ; resume the process     3f6
```

(EPENTER) Global code for entering a process via one of its entry ports. 3g

The form of EP'LOAD is the following: 3g1

```
   EP'LOAD: MOVE     0,REG'BASE+17                              3g1a

            ....                                                3g1b

            ....                                                3g1c

            MOVE     S,STK'PTR(D)   ;load register 16           3g1d

            MOVE     F,FRAME(D)     ;load register 17           3g1e

            MOVE     0,B          ;save aside RecentEp          3g1f

            EXCH     0,RECENT'EP(D)                             3g1g

            MOVEM     0,SAVED'RECENT(D)                         3g1h

            MOVE     0,0(B)     ;save entry-port connection     3g1i

            MOVEM     0,SAVED'CNCTN(D)                          3g1j

            JRST     @ENTRY'POINT(B); start the process         3g1k
```

A process may pass a reference to a port (hereafter called a "ref port" a la ALGOL 68) to a procedure (internal or external) which will perform port calls for it. Since the port indicates by its dseg'ptr to which process it belongs, information must be saved in the dseg when the port is used so that control can get back to the procedure correctly. Since the process's pc is saved on the stack by a procedure call, the procedure can save its pc in the normal PC slot of the calling process's dseg when it makes a port call for the process. 3h

The process may also use ref port variables when doing port calls itself. If the ref port yields a port which belongs to the process attempting to use it, there is no problem: only one thread of control exists, and the process's pc can be saved in the normal way. If, however, the ref port yields a port which does not belong to the process attempting to use it, an error occurs. 3i

A port is inextricably tied to some dseg (and therefore to a specific instance of a particular process) and using it from a different process is inconsistent with that notion since it would be necessary to somehow store knowledge of two separate processes with the port as well as two message buffers, and two different connection words -- in short two distinct ports under the same roof. 3i1

The effect of such usage could be obtained by allowing port variables: a process which wanted a copy of some port to which it had access (by means of a ref port variable) could then "copy" the other port into the variable port. Only the connection information would actually be copied into the port variable; its message buffer, startup cell, and most importantly, its dseg'ptr would be constant just as for a non-variable port in the same process. 3i2

The following code handles port calls from within an external procedure. It saves the linkage and code bases (packed into one word just like LINK'CODE in the DSEG) on the stack and retrieves them from the stack when it regains control after a port call. 3j

There are two possible forms of the code: 3j1

The first uses only in-line code. 3j1a

```
ExtPortCall:MOVE     B,LOCPTR(LNK)        ;save descriptor
for LINK'CODE                                                3j1a1

           PUSH     S,B                                      3j1a2

           MOVE     M,message(D)    ;normal port call code   3j1a3

           MOVE     B,port(D)                                3j1a4

           JSP      P,XPortCall                              3j1a5
```

XPortCall is used instead of PortCall or EPCall because the procedure may not assume that it knows which type it is using, and XPortCall will have to check. 3j1a5a

```
           POP      S,C     ; get linkbase,codebase word     3j1a6

           MOVS     LNK,C                ; and put linkbase into
LNK                                                          3j1a7
```

The alternative has both in-line code and some global code, and is probably the better choice of the two. 3j2

in-line code 3j2a

```
            MOVE    B,port(D)                                     3j2a1

            PUSHJ   S,EXT'PORT'CALL      ;routine to handle such
    port usage                                                    3j2a2

  global code:                                                    3j2b

    EXT'PORT'CALL: MOVE    P,LOCPTR(LNK)                          3j2b1

            PUSH    S,P   ; save his PC value                     3j2b2

            JSP     P,SENDX                                       3j2b3

            POP     S,C   ; restore linkbase and codebase         3j2b4

            MOVS    LNK,C                                         3j2b5

            POPJ    S,                     ; and let the
    external procedure proceed                                    3j2b6
```

The same global routines are used by any port call which uses a ref port since its type cannot be assumed by the in-line code and since the error of using a port in a process which does not own it must be handled. However, the surrounding in-line code which saves and restores the LNK and C registers is only needed when the sender is an external procedure. 3k

A message consists of one word of information. One special value, 400000000000, is designated as the "null message". Thus, a statement such as 3l

```
  PORT port(VariableMessage);                                     3l1
```

may send the null message if VariableMessage has it as its value. If a process attempts to read the message in a port B, it will be told that the port is empty iff it contains the null message. Indeed, whenever a message is read from B, its buffer is marked as containing the null message so that further attempts to read the contents of the buffer will meet with failure. 3m

The code for 3n

```
  [variable '←] "EMPTY" port [signalphrase];                      3n1
```

is the following: 3o

```
          HRLZI    M,400000          ; M ← 400000000000           3o1

          CAMNE    M,port$msg(D)     ; null msg in port?          3o2
```

```
            JRST     movemessage(C)    ; no - contains a valid msg    3o3

             <signalphrase code>                                      3o4

  movemessage: EXCH    M,port$msg(D)      ;mark empty and get msg     3o5

                [MOVEM   M,variable(D) ] - present if [variable
  '←] phrase used                                                     3o6
```

A Felt Need for a Seminar Series

Seminars 1

An important part of a persons augmentation system is what he knows. There is a great deal going on in various corners of this project which it would probably be useful for a wider group to understand in the interest of personal development, project integration and greater flexibility of task allocation. 1a

I do not know who should be responsible for getting a seminar series started but it seems logically to belong to one of the three coordinators. 1b

From my experience in having such a seminar series running at Shell the cost in preparation is more than repaid in the increase in knowledge and understanding of the group and it usually helps the person giving the seminar in organizing his thoughts and in obtaining useful feedback from the rest of the group. 1c

The list of subjects needing discussion is long. I would recommend we start with a series on Tenex before Ken leaves. 1d

Possible Topics 1e

Tenex Ken Don A. Don W. John 1e1

NLS Bill Charles Mimi 1e2

Modular programming system Bill 1e3

Property list stuff Bill 1e4

Output Processor Bruce Walter 1e5

Journal Bill Harvey 1e6

DEX Harvey Doug 1e7

Collector-sorter Bill 1e8

Aspects of the hardware Roger ED others 1e9

Network Protocols John Dick 1e10

Treemeta Don 1e11

L 10 Bill 1e12

Baseline system Jim Bruce 1e13

Catalog system Dick Jim 1e14

ETC 1e15

A Felt Need for a Seminar Series

(J7343) 23-JUN-71 13:19; (Expedite) Title: Author(s): Richard W. Watson/RWW; Distribution: Charles H. Irby, William H. Paxton, Bruce L. Parsley, William S. Duvall, Mimi S. Church, John T. Melvin, Kenneth E. Victor, Walter L. Bass, Ed K. Van De Riet, Douglas C. Engelbart, James C. Norton, Harvey G. Lehtman, J. D. Hopper, Don C. Wallace, Kenneth E. Victor/CHI WHP BLP WSD MSC JTM KEV WLB EKV DCE JCN HGL JDH DCW KEV; Sub-Collections: ARC; Clerk: RWW;

More on NLS Error Messages

Further Note on NLS Errors. 1

In re-organising somm of the lower level file routines, I noticed tht some error messages were deleted, specifically those in openpc. 1a

Seemigly, they were deleted in favor of ones produced by lower level routines. 1b

I personally feel that error messages emitted at low levels should be overidden at higher levels, if the routines at higher levels have a better awareness of te meaning of the error in the user context. 1c

Such is the case here, where 'No Such Version' means much less to the user than the message 'PC does not exist'. 1d

I would like to restore the error messages to openpc, and subsequently adopt the philosophy of emitting error messages which are meaningful in the users context wherever poissible. 1e

More on NLS Error Messages

(J7344) 24-JUN-71 21:58; Title: Author(s): William S. Duvall/WSD; Distribution: Mimi S. Church, Charles H. Irby, Bruce L. Parsley, Walter L. Bass, William H. Paxton/MSC CHI BLP WLB WHP; Sub-Collections: ARC; Clerk: WSD;

WSD 28-JUN-71 15:01 7345

Functional Description Of Groups in the Identification System

This is derived largely from (6215,)

WSD 28-JUN-71 15:01 7345

Functional Description Of Groups in the Identification System

Group Identification 1

General Description 1a

The identification for a group is identical in form to that for a person. 1a1

Syntax: L $LD 1a1a

At the level of the user typing in a identification list, there is normally no distinction. 1a1b

There is, however, one exception. 1a1c

A group may be referenced in one of three manners. 1a1c1

Expanded References. 1a1c1a

When a group identification is being used as a substitute for the identifications of the individuals belonging to that group, the reference is said to be expanded. 1a1c1a1

This is indicated syntactally by preceding the identification by the chracter '↑, e.g. ↑DSSIG is an expanded reference to the DSSIG group. 1a1c1a2

Un-expanded references 1a1c1b

There may be instances where the desire is to reference the group itself as an entity, rather than the members of the group. 1a1c1b1

This is an un-expanded reference, and is indicated by preceding the identification with the character '&, e.g. &DSSIG. 1a1c1b2

The character '& is chosen due to a relatively weak similarity of this function to the REF variables in L10. 1a1c1b2a

Normal (Default) References 1a1c1c

When the identification of a group is used alone, e.g. DSSIG, it will be expanded or not depending on the information contained in the identification record for the group. 1a1c1c1

Group references should normally be made in this manner. 1a1c1c2

Modification to the Identification Record Format 1b

Syntax: '(<identification> ') ["Expand"] "Group ("
<identification list> ') $NP <Proper name> <affilitaion>
<Mailing address> EOL EOL
<Comments> 1b1

The optional "Expand" parameter specifies whether normal references to the group are treated as expanded or un-expanded references. 1b2

The default setting will be to expand. 1b2a

The identification list following the word 'Group' describes the membership of the group. 1b3

Note that the identification list may include: 1b3a

(1) Identifications of people 1b3a1

(2) Identifications of other groups (as normal, expanded or un-expanded references) 1b3a2

An expanded reference to another group is expanded if and only if the reference to he current group was expanded. 1b3a2a

(3) Comments 1b3a3

The proper name is the full name of the group, e.g. Dialogue Support System Interest Group. 1b4

The address field contains a mailing address for un-expanded references to the group. 1b5

This would presumably be a secretary, coordinator, etc. 1b5a

The identification of some user may be used in lieu of an actual address. 1b5b

Example 1b6

(DSSIG) Expand Group (wsd msc dce chi hgl jcn blp whp
rww) Dialogue Support System Interest Group ARC
WSD

User: JOURNAL;
Sub-Collections: ARC;
Delivery: Hard Copy; 1b6a

Changes to Identification Lists 1c

The only change which the inclusion of group identifications in identification lists brings is the inclusion of the expanded and un-expanded reference operators, '↑ and '&. 1c1

As expounded elsewhere, these characters signify that references to a group are to be expanded or un-expanded(regardless of the expand parameter in the group identification record). 1c2

The presence of these characters preceding a pesonal identification is an error condition, and ignored. 1c3

The identification of the individual in this case is included in the identificaton list. 1c3a

Examples: &DSSIG &NICIG ↑DSSIG ↑NICIG 1c4

(J7345) 28-JUN-71 15:01; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Marilyn F. Auerbach, Mimi S. Church, Charles H. Irby, Harvey G. Lehtman, Richard W. Watson/MFA MSC CHI HGL RWW; Keywords: Identification Groups; Sub-Collections: ARC; Clerk: WSD;

WSD 28-JUN-71 15:07 7346

Syntax and Semantics of TNLS Identification Sub-mode

See (7345,) for information relating to Group Identification

Identification NLS Submode 1

This section describes the syntax and semantics of the commands in the TNLS identification submode. 1a

The syntax and semantics of commands in the DNLS submode will presumably be similar. 1a1

General Description 1b

The Identification Submode may be entered either directly from the TNLS command level, or--for the purpose of entering a new user--from entering an identification list within some nls command. 1b1

Some of the information in an identification record should not be changed by ordinary users. 1b2

Consequently, two levels of protection are allowed. 1b2a

(1) Enabled NLS user. 1b2a1

An enable/disable mechanism will be provided in NLS whereby a user may gain access to certain commands by an enable command. 1b2a1a

In order to enable ones status, the appropriate fields must be set in his identification record. 1b2a1a1

(2) Password access commands. 1b2a2

Certain commands, such as delete user, are sufficiently dangerous that a user must be enabled and provide a password in order to execute them. 1b2a2a

Three basic capabilities will be allowed by the identification submode. 1b3

(1) Enter New Identification. 1b3a

(2) Modify existing identification 1b3b

(3) Delete Identification. 1b3c

When entering the identification submode from TNLS, either of the three command sets may be invoked by typing 'E[nter], 'M[odify] or 'D[elete]. 1b4

If the submode is entered from the identification list level, however, the user is automatically placed into the enter mode followed by the modify mode. 1b5

After the modify mode is exited, the system returns a value equal to the identification of the new user, and control returns to the identification list parser. 1b6

Commands 1c

Identification Sub-mode Entry 1c1

(a) From TNLS 1c1a

E[xecute] ID[entification Sub-mode] CA. 1c1a1

This command will cause te user to be plaed in the I.D Sub-mode. 1c1a1a

TNLS will respond with the hearald character '>. 1c1a1b

The user may then proceed with any legal I.D. Submode commands. 1c1a1c

After each command is successfully completed. and after all CD's and errors, he will return to this level until he executes a Quit command. 1c1a1d

(b) From an Identification List 1c1b

A CR typed in an identifition list causes entry to the Identification Submode. 1c1b1

TNLS responds to the CR as though it were the 'E for the Enter command. 1c1b2

When the Enter Command has been completed, the entry is typed to the user, and the Modify command is entered. 1c1b3

When the Modify has been completed, the string value of the new user is returned to the identification list parser. 1c1b4

Any errors or command deletes from this level cause a null string to be returned to the identification list parser. 1c1b5

Enter Command. 1c2

Syntax: E[nter Identification for] (I[ndividual] / CA
[Individual] / G[roup]) [
Name:] LITERAL CA [
Address:] (LITERAL /IDENT) CA [
Affiliation:] LITERAL CA [
(if Group) Membership:] IDENTLIST CA [
Identification:] (LITERAL CA/ CA) 1c2a

Semantics: 1c2b

E[nter Identificationn for] (I[ndividual] / CA
[Individual] /G[roup]) 1c2b1

This specifies whether the new identiication is to be for an individual or group. 1c2b1a

[CR Name:] LITERAL CA 1c2b2

This is either the full name of the individual, or the Proper nnme name of the group. 1c2b2a

In the case of individuals, the identification file is searched for entries with the same last name. If any are found, the corresponding entries are typed to the user, and he he is asked to respond yes or no as to whether that person is the intended entry. 1c2b2b

In the event of an affirmative response, the command is terminated. 1c2b2b1

For Groups, a slightly more complicated search is done, where the proper names of groups in the identification file are compared to the proper name offered, and suitable interaction takes place if they are similar. 1c2b2c

[CR Address:] (LITERAL/IDENT) CA 1c2b3

This is the mailing address for the entry. 1c2b3a

In the case of indals, it must be a normal, textual mailng address. 1c2b3b

For Groups, it may either be a normal mailing address, or an IDENT of some recognised user or group. 1c2b3c

If it is the ident of a group, it may be preceded by an expanded or un-expanded reference command, or it may be a normal reference. 1c2b3c1

References to other groups as mailing addresses are handled in the obvious manner. 1c2b3c2

If an illegal IDENT is supplied, the user is asked to re-enter the field. 1c2b3d

[CR Affiliation:] LITERAL CA 1c2b4

This is the Professional affiliation of the new user, e.g. ARC or UCLA. 1c2b4a

If the LITERAL is empty, then an affiliation of "INDEPENDENT" is used. 1c2b4b

For Groups, the affiliation should indicate the Professional entity with which that groups activities are based, e.g. ARC, NIC, etc. 1c2b4c

[CR Membership:] IDENTLIST CA 1c2b5

This field is significant only for Groups. 1c2b5a

It is the list of users/groups who make up the initial membership of the group. 1c2b5b

It will be parsed as a normal Identification list, which means that new entries may be made within the list. 1c2b5c

[CR Identification:] (LITERAL CA /CA) 1c2b6

This selects the identification which will be used for te new user being entered. 1c2b6a

If a CA is typed, th system will select the identification according to the following algorithm: 1c2b6b

(1) Make a string of 'Initials' by selecting the first character from each word in the name (where words are separated by spaces). 1c2b6b1

(2) Make a check to see if it is unique. 1c2b6b2

If it is not unique, append a digit to the end (initially 0). 1c2b6b2a

Continue incrementing the value of the digit until a unique string is found. 1c2b6b2b

(3) Return this as the value of the new identification. 1c2b6b3

If a Literal is typed, it is assumed that the literal contains the string to be used for the new users Identification. 1c2b6c

The string is checked for legality (The syntax must be L $LD), and then for uniqueness. 1c2b6c1

IF either check fails, the user is asked to re-enter the field. 1c2b6c2

Modify Command 1c3

Syntax: M[odify record for] IDENT CA 1c3a

Semantics: 1c3b

This command is used to enter the Modify sub-submode. 1c3b1

Assuming the IDENT is legal, the user enters a command level where any Command Deletes or serious errors return him to the Identification submode, and the following commands are legal:. 1c3b2

CONVENTION: 1c3b2a

The term TYPEOLD is used in the descriptions of these commands to mean the following: 1c3b2a1

The old contents of the field are typed to the user. 1c3b2a1a

If the next thing a user types is a CA, the command is treated as a NO-OP, an the command is terminated.. 1c3b2a1b

If there is no explanation of a commands use under the syntax, the semantics of the command are substantially the same as those used under the ENTER command. 1c3b2a2

A[ffiliation:] TYPEOLD LITERAL CA 1c3b2b

D[elivery:] TYPEOLD $('On-Line / 'Hard Copy / LITERAL CA / CA) 1c3b2c

This allows the specification of the default delivery techniques to be used for JOurnal documents directed to this user. 1c3b2c1

On-Line and Hard Copy are the two standard ones currently used, and LITERAL may be used to describe a new one, or one to be meaningful at some future date. 1c3b2c2

The user may specify more than one type of delivery with this command, as it is not terminated until a redundant CA is typed. 1c3b2c3

Once the delivery field has been set up, the user will get delivery of documents only in the manner specified by this field. 1c3b2c4

This means that if he were getting delivery previously in various manners by default (i.e. the field was not there), specification of this field could subtly stop it. 1c3b2c4a

E[xpand Normal References ?] ANSWER 1c3b2d

This simply sets/resets the flag causing normal references to a group to be expanded. 1c3b2d1

An error is executed if the IDENT being modified is not that of a group. 1c3b2d2

G[roup Membership] TYPEOLD $([+] ((A[dd]/ D[elete]) IDENTLIST) / I[nitialise]) CA) 1c3b2e

This command puts the user into a baby submode where he may add and delete persons from the membership, or initialise (reset) it. 1c3b2e1

A reduntant CA is used to exit. 1c3b2e2

I[dentification:] TYPEOLD LITERAL CA 1c3b2f

M[ailing Address:] TYPEOLD (LITERAL / IDENT) CA 1c3b2g

N[ame:] TYPEOLD LITERAL CA 1c3b2h

ST[atus] CA 1c3b2i

This command causes the value of the various fields in the identification record for the ident currently being modified to be typed. 1c3b2i1

The fields typed may (eventually) be culled according to the users 'enabled' status. 1c3b2i2

SU[b-Collections:] TYPEOLD $(A[RC] / N[IC] / LITERAL CA) 1c3b2j

This allows the specification of the subcollections to which Journal items submitted by this user should by default belong. 1c3b2j1

Any number of subcollections may be specified, and a redundant CA is used to terminate the command. 1c3b2j2

U[ser (For TENEX):] USERNAME CA 1c3b2k

This allows the association of the user/group with a TENEX user name. 1c3b2k1

The immediate effect of this will be that any on-line delivery of Journal Documents will b done under the specified TENEX directory. 1c3b2k2

The legality of the username will be checked. 1c3b2k3

Delete Command 1c4

Syntax: D[elete Identification:] IDENT CA [
Password:] PASSWORD [
(type out of I record)
OK??] ANSWER 1c4a

Semantics: 1c4b

This allows identification records to be deleted. 1c4b1

In order to use this command, a user must be enabled, and he must know the password. 1c4b2

The identification record of the prospective deletee is typed before a final affirmation to help avoid mistakes. 1c4b3

Syntax and Semantics of TNLS Identification Sub-mode

(J7346) 28-JUN-71 15:07; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Marilyn F. Auerbach, Mimi S. Church, Charles H. Irby, Harvey G. Lehtman, Richard W. Watson/MFA MSC CHI HGL RWW; Keywords: Identification Sub-mode Syntax Semantics; Sub-Collections: ARC; Clerk: WSD;

Schedule

Schedule is now in the Journal (#7261) and posted near the blackboard in the Console Room.

EKV

Schedule

(J7347) 28-JUN-71 15:42; (Expedite) Title: Author(s): Ed K. Van De Riet/EKV; Distribution: Marilyn F. Auerbach, Walter L. Bass, Roger D. Bates, Mimi S. Church, William S. Duvall, Beauregard A. Hardeman, Martin E. Hardy, Fred P. Hocker, J. D. Hopper, Charles H. Irby, Mil Jernigan, Harvey G. Lehtman, John T. Melvin, Jeanne B. North, James C. Norton, Cindy Page, Bruce L. Parsley, William H. Paxton, Jeffrey C. Peters, Barbara E. Row, Ed K. Van De Riet, Ed K. Van De Riet, Dirk H. van Nouhuys, Kenneth E. Victor, Don C. Wallace, Richard W. Watson, Don I. Andrews/MFA WLB RDB MSC WSD BAH MEH FPH JDH CHI MEJ HGL JTM JBN JCN CXP BLP WHP JCP BER EKV EKV DVN KEV DCW RWW DIA; Sub-Collections: ARC; Clerk: BER;
Origin: <ROW>BLANK.NLS;1, 28-JUN-71 14:42 BER ;

WSD 28-JUN-71 16:14 7349

Possibilities for improvement of Journal Delivery

Please let me know if you see anything in here (or can think of anything not in here) which you would like to see in a not-too-extensive upgrading of Journal Delivery.

WSD 28-JUN-71 16:14 7349

Possibilities for improvement of Journal Delivery

Possible enhancements for Journal On-Line Delivery 1

Use more sophistication in determining which documents should be delivered on-line versus hard copy, and to whom. 1a

The sender should be able to specify that a document absulutely should be delivered as Hard-copy/On-line. 1a1

JCN feels strongly about this one, but I am still not quite convinced that it is necessary----almost, but not quite. 1a1a

I guess that a sender should be able to say "This document is not worth getting hard copy of" or conversely "This is an important document, and everyone should recieve hard copy of it". 1a1b

OK...I guss mebbee I am convinced. 1a1c

Perhaps we should have the ability to treat messages and Documents separately, e.g. messages on-line only, and documents both. 1a2

The recipient of a document should be able to easily request the supression or printing of hard copy for some document he has recieved. 1a3

Upon seeing a document in his control file, he should be able to say "Print That" or "Don't print that, I've seen it". 1a3a

The delivery method to be used for documents/messages should be settable by source as well as destination. 1a4

The user should be able to say: "I want all documents from XXX to be delivered to me in Hard Copy only" 1a4a

Allow alternate destinations for documents. 1b

The user should, for example, be able to say: "All documents addressed to me in the sub-collection NIC I want delivered on-line to me with a hard copy to XXX". 1b1

Alternatively, he should be able to direct documents from certain sources to different persons, e.g. secretaries. 1b2

This could concievably be used to provide an automatic culling facility, e.g. suppose that documents could be directed to particular sets depending on their source, sub-collection membership, keywords, etc. 1b3

It would then be simple for a user to keep updated sets of documents, without spending a great deal of manual effort sorting them. 1b3a

Allow on-line delivery of Author Copies. 1c

This came from a suggestion from RWW. 1c1

I think that mebbe they should be put in a separate branch at the authors option. 1c2

(J7349) 28-JUN-71 16:14; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Charles H. Irby, Harvey G. Lehtman, Jeanne B. North, James C. Norton, Bruce L. Parsley, Richard W. Watson, Dirk H. van Nouhuys/CHI HGL JBN JCN BLP (This should go into the Needs/Possibilities file) RWW DVN; Keywords: Journal Delivery Needs Possibilities; Sub-Collections: ARC; Clerk: WSD;

Notes from File Space Meeting

SMALL MEETING CONCERNING FILE SPACE 1

Technical type things 1a

Melvin Norton Van de Riet Wallace Watson 1a1

Attendees 1b

The Basic problem seems to be that the system simply does not have enough storage capacity to satisfy the current and immediate future ARC/NIC requirements 1b1

We assume that this type of situation will probably always be so i.e. our file appetites are continuously growing, thus whatever approach is taken will have to be practical and flexible 1b2

The solutions seem to be administrative, their implemenataion technical 1b3

Ed would be the person responsible for the administrative things 1b4

Administrative type things 1b4a

there need to be better types of summaries and reports of disc usage etc. 1b4a1

limits could be set on how much space any one user or group of users could have 1b4a2

Discussed 1b4b

the system does not lend itself to minimizing number of files user may generate and leave around 1b4b1

the output processor could, for example, take away the option of naming the output file and use one filename, extension, and version for its output 1b4b1a

greater use of temporary files could be made (as in the 940)? 1b4b1b

we must get some sort of backup system into operation 1b4b2

it should be as fully automated as possible but an interim system should be devised if necessary involving the use of an operator or some sort of manual mechanism 1b4b2a implementation of some sort of administratively defined limits 1b4b3

GTJFN could give a fail return if user is over his limit 1b4b3a

the EXEC could require that the user do something about his files prior to letting him login or logout 1b4b3b

Notes from File Space Meeting

(J7350) 28-JUN-71 16:20; Title: Author(s): John T. Melvin/JTM; Sub-Collections: ARC; Clerk: WSD;

WSD 28-JUN-71 16:27 7351

Comments on File Space meeting notes (7350,)

With regard to memo on file space meeting (7350,) 1

If the Output Processor were to take away the option of naming output files, it would make things very hard for Journal Hard Copy Delivery, i.e. a lot of chnges would need be made, and things could not be done as effeciently as they are now. 1a

I like the idea of using more temporary files. 1b

I think that the exec should check on file space usage at some inocuous point. 1c

Bombing out of a getjfn can cause nasty problems for suffering programs which are already trying as hard as they can to cope with the file system. 1c1

Mebee the EXEC could check file space usage at reset time??? 1c2

Comments on File Space meeting notes (7350,)

(J7351) 28-JUN-71 16:27; Title: Author(s): William S. Duvall/WSD; Distribution: John T. Melvin, James C. Norton, Ed K. Van De Riet, Richard W. Watson, Don C. Wallace/JTM JCN EKV RWW DCW; Keywords: File Space; Sub-Collections: ARC; Clerk: WSD;

JCN 29-JUN-71 9:24 7356

Note to Duane Stone

JCN 29-JUN-71 9:24 7356

Note to Duane Stone

Thanksfor the message last week. We hope your return trip went well. I note you worked online on June 24th. Did you use the Execuport, and if so, did you use lowercase mode when in TNLS? If you do not plan to use the Model 37 to connect, I'll take the permanent 15cps switch off...OK? 1

Note to Duane Stone

(J7356) 29-JUN-71 9:24; Title: Author(s): James C. Norton/JCN; Distribution: Duane L. Stone, Richard W. Watson/DLS (Note the entry in your initial file..try sending one to me?) RWW; Keywords: ; Sub-Collections: ARC; Clerk: JCN;

WSD 29-JUN-71 13:39 7358

NLS Identification Submode (Version II)

This supercedes the previous version (7251,), the major change being the addition of the capabilities sub-command in modify.

Identification NLS Submode 1

This section describes the syntax and semantics of the commands in the TNLS identification submode. 1a

The syntax and semantics of commands in the DNLS submode will presumably be similar. 1a1

General Description 1b

The Identification Submode may be entered either directly from the TNLS command level, or--for the purpose of entering a new user--from entering an identification list within some nls command. 1b1

Some of the information in an identification record should not be changed by ordinary users. 1b2

Consequently, two levels of protection are allowed. 1b2a

(1) Enabled NLS user. 1b2a1

An enable/disable mechanism will be provided in NLS whereby a user may gain access to certain commands by an enable command. 1b2a1a

In order to enable ones status, the appropriate fields must be set in his identification record. 1b2a1a1

(2) Password access commands. 1b2a2

Certain commands, such as delete user, are sufficiently dangerous that a user must be enabled and provide a password in order to execute them. 1b2a2a

Three basic capabilities will be allowed by the identification submode. 1b3

(1) Enter New Identification. 1b3a

(2) Modify existing identification 1b3b

(3) Delete Identification. 1b3c

When entering the identification submode from TNLS, either of the three command sets may be invoked by typing 'E[nter], 'M[odify] or 'D[elete]. 1b4

If the submode is entered from the identification list level, however, the user is automatically placed into the enter mode followed by the modify mode. 1b5

After the modify mode is exited, the system returns a value equal to the identification of the new user, and control returns to the identification list parser. 1b6

Commands 1c

Identification Sub-mode Entry 1c1

(a) From TNLS 1c1a

E[xecute] ID[entification Sub-mode] CA. 1c1a1

This command will cause te user to be plaed in the I.D Sub-mode. 1c1a1a

TNLS will respond with the hearald character '>. 1c1a1b

The user may then proceed with any legal I.D. Submode commands. 1c1a1c

After each command is successfully completed. and after all CD's and errors, he will return to this level until he executes a Quit command. 1c1a1d

(b) From an Identification List 1c1b

A CR typed in an identifition list causes entry to the Identification Submode. 1c1b1

TNLS responds to the CR as though it were the 'E for the Enter command. 1c1b2

When the Enter Command has been completed, the entry is typed to the user, and the Modify command is entered. 1c1b3

When the Modify has been completed, the string value of the new user is returned to the identification list parser. 1c1b4

Any errors or command deletes from this level cause a null string to be returned to the identification list parser. 1c1b5

When the user is rturned to the identification list parser, a message reflecting the status is typed to the user. 1c1b6

Enter Command. 1c2

```
Syntax: E[nter Identification for ] (I[ndividual] / CA
[Individual] / G[roup]) [
Name: ] LITERAL CA [
Address: ] (LITERAL /IDENT) CA [
Affiliation: ] LITERAL CA [
(if Group) Membership: ] IDENTLIST CA [
Identification: ] (LITERAL CA/ CA)                         1c2a
```

Semantics: 1c2b

E[nter Identificationn for] (I[ndividual] / CA [Individual] /G[roup]) 1c2b1

This specifies whether the new identiication is to be for an individual or group. 1c2b1a

[CR Name:] LITERAL CA 1c2b2

This is either the full name of the individual, or the Proper nnme name of the group. 1c2b2a

In the case of individuals, the identification file is searched for entries with the same last name. If any are found, the corresponding entries are typed to the user, and he he is asked to respond yes or no as to whether that person is the intended entry. 1c2b2b

In the event of an affirmative response, the command is terminated. 1c2b2b1

For Groups, a slightly more complicated search is done, where the proper names of groups in the identification file are compared to the proper name offered, and suitable interaction takes place if they are similar. 1c2b2c

[CR Address:] (LITERAL/IDENT) CA 1c2b3

This is the mailing address for the entry. 1c2b3a

In the case of indals, it must be a normal, textual mailng address. 1c2b3b

For Groups, it may either be a normal mailing address, or an IDENT of some recognised user or group. 1c2b3c

If it is the ident of a group, it may be preceded by an expanded or un-expanded reference command, or it may be a normal reference. 1c2b3c1

References to other groups as mailing addresses are handled in the obvious manner. 1c2b3c2

If an illegal IDENT is supplied, the user is asked to re-enter the field. 1c2b3d

[CR Affiliation:] LITERAL CA 1c2b4

This is the Professional affiliation of the new user, e.g. ARC or UCLA. 1c2b4a

If the LITERAL is empty, then an affiliation of "INDEPENDENT" is used. 1c2b4b

For Groups, the affiliation should indicate the Professional entity with which that groups activities are based, e.g. ARC, NIC, etc. 1c2b4c

[CR Membership:] IDENTLIST CA 1c2b5

This field is significant only for Groups. 1c2b5a

It is the list of users/groups who make up the initial membership of the group. 1c2b5b

It will be parsed as a normal Identification list, which means that new entries may be made within the list. 1c2b5c

[CR Identification:] (LITERAL CA /CA) 1c2b6

This selects the identification which will be used for te new user being entered. 1c2b6a

If a CA is typed, th system will select the identification according to the following algorithm: 1c2b6b

(1) Make a string of 'Initials' by selecting

the first character from each word in the name (where words are separated by spaces). 1c2b6b1

(2) Make a check to see if it is unique. 1c2b6b2

If it is not unique, append a digit to the end (initially 0). 1c2b6b2a

Continue incrementing the value of the digit until a unique string is found. 1c2b6b2b

(3) Return this as the value of the new identification. 1c2b6b3

If a Literal is typed, it is assumed that the literal contains the string to be used for the new users Identification. 1c2b6c

The string is checked for legality (The syntax must be L $LD), and then for uniqueness. 1c2b6c1

IF either check fails, the user is asked to re-enter the field. 1c2b6c2

Modify Command 1c3

Syntax: M[odify record for] IDENT CA 1c3a

Semantics: 1c3b

This command is used to enter the Modify sub-submode. 1c3b1

Assuming the IDENT is legal, the user enters a command level where any Command Deletes or serious errors return him to the Identification submode, and the following commands are legal:. 1c3b2

CONVENTION: 1c3b2a

The term TYPEOLD is used in the descriptions of these commands to mean the following: 1c3b2a1

The old contents of the field are typed to the user. 1c3b2a1a

If the next thing a user types is a CA, the command is treated as a NO-OP, an the command is terminated.. 1c3b2a1b

If there is no explanation of a commands use under the syntax, the semantics of the command are substantially the same as those used under the ENTER command. 1c3b2a2

A[ffiliation:] TYPEOLD LITERAL CA 1c3b2b

C[apabilities:] TYPEOLD $(N[LS] / E[nable] / LITERAL CA / CA) 1c3b2c

This command allows specification of the capbilities the user has when using the system. 1c3b2c1

The two currently defined capabilities are: 1c3b2c2

NLS. This user may use the NLS system 1c3b2c2a

ENABLE. This reflects y Enable his status to use priveledged commands 1c3b2c2b

D[elivery:] TYPEOLD $('On-Line / 'Hard Copy / LITERAL CA / CA) 1c3b2d

This allows the specification of the default delivery techniques to be used for JOurnal documents directed to this user. 1c3b2d1

On-Line and Hard Copy are the two standard ones currently used, and LITERAL may be used to describe a new one, or one to be meaningful at some future date. 1c3b2d2

The user may specify more than one type of delivery with this command, as it is not terminated until a redundant CA is typed. 1c3b2d3

Once the delivery field has been set up, the user will get delivery of documents only in the manner specified by this field. 1c3b2d4

This means that if he were getting delivery previously in various manners by default (i.e. the field was not there), specification of this field could subtly stop it. 1c3b2d4a

E[xpand Normal References ?] ANSWER 1c3b2e

This simply sets/resets the flag causing normal references to a group to be expanded. 1c3b2e1

An error is executed if the IDENT being modified is not that of a group. 1c3b2e2

G[roup Membership] TYPEOLD $(([
+] ((A[dd]/ D[elete]) IDENTLIST) / I[nitialise])
CA) 1c3b2f

This command puts the user into a baby submode where he may add and delete persons from the membership, or initialise (reset) it. 1c3b2f1

A reduntant CA is used to exit. 1c3b2f2

I[dentification:] TYPEOLD LITERAL CA 1c3b2g

M[ailing Address:] TYPEOLD (LITERAL / IDENT) CA 1c3b2h

N[ame:] TYPEOLD LITERAL CA 1c3b2i

ST[atus] CA 1c3b2j

This command causes the value of the various fields in the identification record for the ident currently being modified to be typed. 1c3b2j1

The fields typed may (eventually) be culled according to the users 'enabled' status. 1c3b2j2

SU[b-Collections:] TYPEOLD $(A[RC] / N[IC] /
LITERAL CA) 1c3b2k

This allows the specification of the subcollections to which Journal items submitted by this user should by default belong. 1c3b2k1

Any number of subcollections may be specified, and a redundant CA is used to terminate the command. 1c3b2k2

U[ser (For TENEX):] USERNAME CA 1c3b2l

This allows the association of the user/group with a TENEX user name. 1c3b2l1

The immediate effect of this will be that any on-line delivery of Journal Documents will b done under the specified TENEX directory. 1c3b2l2

The legality of the username will be checked. 1c3b2l3

Delete Command 1c4

```
Syntax: D[ elete Identification: ] IDENT CA [
Password: ] PASSWORD [
(type out of I record)
OK?? ] ANSWER                                    1c4a
```

Semantics: 1c4b

This allows identification records to be deleted. 1c4b1

In order to use this command, a user must be enabled, and he must know the password. 1c4b2

The identification record of the prospective deletee is typed before a final affirmation to help avoid mistakes. 1c4b3

NLS Identification Submode (Version II)

(J7358) 29-JUN-71 13:39; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Mimi S. Church, Harvey G. Lehtman, Charles H. Irby, Marilyn F. Auerbach, James C. Norton, Richard W. Watson/MSC HGL CHI MFA JCN RWW; Keywords: Identificaion Submode NLS; Sub-Collections: ARC; Clerk: WSD;

(J7359) 29-JUN-71 15:10; (Expedite) Title: Author(s): James G. Mitchell/JGM; Distribution: James G. Mitchell, William H. Paxton, Butler W. Lampson, Alan C. Kay, L. Peter Deutsch/JGM WHP BWL ACK LPD; Keywords: ; Sub-Collections: NIC; Clerk: JGM;
Origin: <MITCHELL>PROCESSES.NLS;12, 29-JUN-71 14:50 JGM ;

own it must be handled. However, the surrounding in-line code which saves and restores the LNK and C registers is only needed when the sender is an external procedure. 3k

A message consists of one word of information. One special value, 400000000000, is designated as the "null message". Thus, a statement such as 3l

```
  PORT port(VariableMessage);                                    3l1
```

may send the null message if VariableMessage has it as its value. If a process attempts to read the message in a port B, it will be told that the port is empty iff it contains the null message. Indeed, whenever a message is read from B, its buffer is marked as containing the null message so that further attempts to read the contents of the buffer will meet with failure. 3m

The code for 3n

```
  [variable '←] "EMPTY" port [signalphrase];                     3n1
```

is the following: 3o

```
          HRLZI    M,400000           ; M ← 400000000000          3o1

          CAMNE    M,port$msg(D)      ; null msg in port?         3o2

          JRST     movemessage(C)     ; no - contains a valid msg 3o3

            <signalphrase code>                                   3o4

  movemessage: EXCH     M,port$msg(D)      ;mark empty and get msg 3o5

              [MOVEM   M,variable(D) ] - present if [variable
  '←] phrase used                                                 3o6
```

The first uses only in-line code. 3j1a

```
  ExtPortCall:MOVE    B,LOCPTR(LNK)        ;save descriptor
  for BASES                                                  3j1a1

            PUSH    S,B                                      3j1a2

            MOVE    M,message(D)    ;normal port call code   3j1a3

            MOVE    B,port(D)                                3j1a4

            JSP     P,XPortCall                              3j1a5
```

XPortCall is used instead of PortCall or EPCall because the procedure may not assume that it knows which type it is using, and XPortCall will have to check. 3j1a5a

```
            POP     S,C      ; get linkbase,codebase word    3j1a6

            MOVS    LNK,C                 ; and put linkbase into
  LNK                                                        3j1a7
```

The alternative has both in-line code and some global code, and is probably the better choice of the two. 3j2

in-line code 3j2a

```
            MOVE    B,port(D)                                3j2a1

            PUSHJ   S,EXT'PORT'CALL     ;routine to handle such
  port usage                                                 3j2a2
```

global code: 3j2b

```
  EXT'PORT'CALL: MOVE    P,LOCPTR(LNK)                       3j2b1

            PUSH    S,P   ; save his PC value                3j2b2

            JSP     P,SENDX                                  3j2b3

            POP     S,C   ; restore linkbase and codebase    3j2b4

            MOVS    LNK,C                                    3j2b5

            POPJ    S,                    ; and let the
  external procedure proceed                                 3j2b6
```

The same global routines are used by any port call which uses a ref port since its type cannot be assumed by the in-line code and since the error of using a port in a process which does not

```
          MOVEM    0,SAVED'RECENT(D)                              3g1h

          MOVE     0,0(B)      ;save entry-port connection        3g1i

          MOVEM    0,SAVED'CNCTN(D)                               3g1j

          JRST     @ENTRY'POINT(B); start the process             3g1k
```

A process may pass a reference to a port (hereafter called a "ref port" a la ALGOL 68) to a procedure (internal or external) which will perform port calls for it. Since the port indicates by its dseg'ptr to which process it belongs, information must be saved in the dseg when the port is used so that control can get back to the procedure correctly. Since the process's pc is saved on the stack by a procedure call, the procedure can save its pc in the normal PC slot of the calling process's dseg when it makes a port call for the process. 3h

The process may also use ref port variables when doing port calls itself. If the ref port yields a port which belongs to the process attempting to use it, there is no problem: only one thread of control exists, and the process's pc can be saved in the normal way. If, however, the ref port yields a port which does not belong to the process attempting to use it, an error occurs. 3i

A port is inextricably tied to some dseg (and therefore to a specific instance of a particular process) and using it from a different process is inconsistent with that notion since it would be necessary to somehow store knowledge of two separate processes with the port as well as two message buffers, and two different connection words -- in short two distinct ports under the same roof. 3i1

The effect of such usage could be obtained by allowing port variables: a process which wanted a copy of some port to which it had access (by means of a ref port variable) could then "copy" the other port into the variable port. Only the connection information would actually be copied into the port variable; its message buffer, startup cell, and most importantly, its dseg'ptr would be constant just as for a non-variable port in the same process. 3i2

The following code handles port calls from within an external procedure. It saves the linkage and code bases (packed into one word just like BASES in the DSEG) on the stack and retrieves them from the stack when it regains control after a port call. 3j

There are two possible forms of the code: 3j1 load'base above. If the process needs to have registers i through 17 restored before it resumes execution, each normal port will have 3c

```
            JRST load'base+i                                         3c1
```

in its startup word. If the process has no base registers other than its stack pointer and frame pointers, it will use load'base+16; 3d

In general, if the process requires i base registers, they must be registers 17,...,17-i+1. These registers are laid out in the process's DSEG in the order F,S,13,...,17-i+1, and only as many words as are necessary need be reserved in the DSEG. Also, since this region is variable, it is the last part of the DSEG which must be present for every process; everything in front of it is fixed in size. 3e

The routine load'base has the following form : 3f

```
  load'base+i:  MOVE    i,REG'BASE+17-1(D) ; load register i       3f1

                ...                                                3f2

                ...                                                3f3

                MOVE    S,STK'PTR(D)  ;load'base+16                3f4

                MOVE    F,FRAME(D)    ; the last base register     3f5

                JRST ,  @PC(D)        ; resume the process         3f6
```

(EPENTER) Global code for entering a process via one of its entry ports. 3g

The form of EP'LOAD is the following: 3g1

```
    EP'LOAD: MOVE     0,REG'BASE+17                                 3g1a

             ....                                                   3g1b

             ....                                                   3g1c

             MOVE     S,STK'PTR(D)   ;load register 16              3g1d

             MOVE     F,FRAME(D)     ;load register 17              3g1e

             MOVE     0,B            ;save aside RecentEp           3g1f

             EXCH     0,RECENT'EP(D)                                3g1g
```

port layout: (see also examples in dseg layout above, esp. FAULTPORT) 3b3

port: XWD port,object'port 3b3a

If the port is not connected, object'port is replaced by a a pointer to a "fake" system port called pf'port which will cause control to enter a port-fault error routine using the normal port call machinery to get there. 3b3a1

The port may also be specified as an ignored port: any uses of it act as null operations. This is handled by joining the port to itself: then any use of the port simply causes the process which is shutting down to be immediately resumed. 3b3a2

msg: WORD ;message word 3b3b

dseg'ptr: ADDR DSEG 3b3c

note that this word must be set up for each port in the dseg whenever a copy of the process is created. 3b3c1

startup: JRST @PC(D) ;if process has no base registers at all 3b3d

Alternates, depending on the process and the port, are the following: 3b3d1

Normal port, process with base registers: 3b3d1a

ZWD load'base+i 3b3d1a1

where load'base is a global routine. If the process only has stack and frame base registers, load'base+2 is used, for instance. 3b3d1a1a

entry port, process with or without base registers: 3b3d1b

JRST epenter+i 3b3d1b1

this is used when the port is an entry port. It also performs the function of load'base+i. epenter+0 is used when the process has no base registers to be loaded. 3b3d1b1a

entry'point: ADDR entry'point'value ; only present for an entry port. 3b3e

The support code for port call involves a system routine called defined as the owner of this process, and is assumed responsible for him. 3a34

Code for: [var '←] "PORT" port ['(message')]; where "port" is a normal port 3b

In-line code: 3b1

```
                HRLZI     M,400000    ; the null message            3b1a

    or             MOVE       M,message(D)  ;if the optional
  (message) phrase is present                                       3b1a1

                MOVE      B,port(D)                                 3b1b

                JSP       P,portcall                                3b1c

  or,             JSP     P,EPCall  ; if port is an entry port      3b1c1
```

global code: 3b2

```
  EPCall:       MOVE     1,SAVED'RECENT ;put saved'recent back      3b2a

                EXCH     1,RECENT'EP  ;into RECENT'EP and put       3b2b

                MOVE     0,SAVED'CNCTN ;SAVED'CNCTN back into       3b2c

                MOVEM     0,port(1) ;port which was pointed at
  by RECENT'EP                                                      3b2d

  portcall:     MOVEM          S,STK'PTR(D) ;save stack pointer
  word                                                              3b2e

                MOVEM          F,FRAME(D)   ;save current frame
  pointer                                                           3b2f

  send'no'stk: MOVEM          P,PC(D)         ;save pc              3b2g

                MOVSM          B,(B)          ; railroad switching  3b2h

                MOVE          D,dseg'ptr(B) ;get pointer to
  object port's dseg                                                3b2i

                MOVE          C,@RETLOC(D)  ;load codebase and
  check for not-in-memory trap                                      3b2j

                JRST          STARTUP(B) ;resume the object
  process                                                           3b2k
```

```
                        XWD        400000,0        ;message word for
the START port                                                       3a17

                        ADDR    DSEG                                 3a18

                        JRST       0,EPENTER                         3a19
```

EPENTER is a system routine which handles control arrival over an entry port. 3a19a

```
                        ADDR    entry'point                          3a20

OWNER:      XWD     0,pf'port  ; process's owner port                3a21

            XWD     400000,0  ;null msg in msg buffer                3a22

            ADDR    DSEG                                             3a23

            JRST    @PC(D)  ; or LOAD'BASE+i if base regs            3a24

*           the process's fault port                                 3a25

(FAULTPORT)                                                          3a26

FAULT:                  XWD        FAULT,owner                       3a27
```

owner represents a pointer to the owner port in the process which owns this process. 3a27a

```
                        XWD        400000,0      ;port's message
buffer                                                               3a28

                        ADDR    DSEG                                 3a29

                        JRST       @PC(D)                            3a30
```

This word distinguishes a normal port from an entry port. The address which it contains is used in the port call mechanism. Cf. (PORTCALL). 3a30a

```
*           storage for the registers                                3a31

REG'BASE:FRAME:         XWD        0,frame'ptr                       3a32

STK'PTR:                XWD        max'stack,stack'ptr               3a33
```

Any other base registers which the process needs to have loaded are placed following STK'PTR 3a33a

The process to which this process's fault port is connected is

(PORTCONTROL) Port Control: Code and Semantics 3

Layout of The Data Segment of a Process 3a

```
DSEG:SEG'NUMBERS: XWD          dsegn,csegn                        3a1

BASES:          XWD          dsegbase,codebase                    3a2

LOCPTR:              ADDR    RETLOC                               3a3

RETLOC:              MOVE         C,BASES                         3a4

          or         MOVE         C,nonxmem                       3a4a

(??)                    MOVS          LNK,C                       3a5

                     JRSTF        @-2(S)                          3a6

PC:                   ADDR    pc'value                            3a7

RECENT'EP:            ADDR     0       ;most recent entry-port over
which control arrived                                             3a8

SAVED'RECENT:        ADDR     0        ;recent'port saved here    3a9

SAVED'CNCTN:       ADDR     0   ;connection for RECENT'EP port    3a10

* the name of this process:                                       3a11

MYNAME:   ASCII    'process name'                                 3a12

          ASCII    'process name'                                 3a13

FAMILY:   XWD      son'list,brother'link                          3a14
```

The son'list pointer points to the most recently acquired son process of this process; that son and all his brother processes are linked in a linear list by the brother'list field in each of their dsegs. Son'list=0 means that this process has no son processes. If brother'link=0, this is the last process on its parent's son list. Both these pointers refer to the beginning of data segments. 3a14a

```
*          the process's start port (an entry port)               3a15

START:               XWD          START,pf'port                   3a16
```

pf'port is a "port" in the system which is used to field port faults. Any unconnected port is, in reality, connnected to the pf'port. 3a16a statement: it is automatically ENABLEd at the start of that statement and CANCELed on its successful completion. 2h3f

Simple Catcher Determination and Actions 2i

A catch-phrase can list a set of specific codes, "classes" of codes or "all codes" on which it is prepared to act. The actions which it may take on a given error or class of errors is one of the following: 2i1

(a) an arbitrary statement. 2i1a

(b) VALUE expression: this action takes the value of the expression as the value of the called procedure and execution of the receiver will continue in the same manner as it would on a normal return from the called procedure. 2i1b

In both cases (a) and (b), before the error action is executed, the call stack is cut back to the same point it would have been at on a normal return to the receiver. 2i1b1

(c) "INVOKE" procedure call: in this case, the call stack remains as it was when the error was generated, and the procedure in the error action is called "almost as if" it had been called by the error generator. 2i1c

Signals Between Processes 2j

Signal Messages across Ports 2j1

No SIGNAL facilities are provided for processes talking to one another across ports (with the exception of the OWNER/FAULT paths). However, since errors can occur in attempting to use a port (connection, or control fault) a catch-phrase can be appended to a port call to field such conditions within the running process. Once generated, such a signal looks like any other and could be fielded by any pocedures in the call hierarchy of the running process. 2j1a

The FAULT-OWNER Chain as a Signal Path 2j2

When any signal is not fielded by a process itself, it is propogated up the FAULT/OWNER chain in an attempt to find someone to accept it. In each process, the signal passes through the same stages that any signal would. When it is finally fielded, that process's OWNER port is JOINed to the FAULT port of the process at which the signal originated. 2j2a

This control scheme is closely analogous to the scheme within a process. 2j2b

A procedure declares itself a candidate signal-catcher by providing a CATCH-phrase (or sequence of CATCH-phrases) which will inspect a generated signal when requested during the backwards scan through the procedure call hierarchy and either accept the signal or reject it. Rejecting it will cause the backwards scan to continue; accepting it allows the CATCH-phrase to take some simple action, after which the normal flow of control will resume in the procedure containing the CATCH-phrase. 2h3b

The syntax of a CATCH-phrase is 2h3b1

```
catchphrase = "CATCH" [lhs] '( $(caserel ': erroraction
';) ');                                                  2h3b1a
```

error actions will be described shortly; caserel means what it normally does in MPL(A), except that the value being compared in each binary relation (caserel) is the signal value. If the optional lhs is present, the value of the signal is assigned to it. 2h3b2

A CATCH-phrase is "provided" as a potential signal catcher either by the execution of an ENABLE statement or by appending the phrase to a statement [and to individual operators in some later version] 2h3c

The ENABLE statement has the syntax: 2h3d

```
[label ':] "ENABLE" (labelid / catchphrase);             2h3d1
```

[can an ENABLE statement have a catch phrase attached to it?] 2h3d1a

The CATCH-phrase enabled is either the one appended to the ENABLE clause or the CATCH-phrase in another ENABLE statement identified by the labelid. When an ENABLE statement is executed during normal execution, the address of the CATCH-phrase is pushed onto a (linked) "CATCH-stack" associated with that incarnation of the procedure. If the CATCH-phrase is already enabled (and therefore already has an entry in the catch-stack), it is first removed from its previous position before being pushed onto the top of the stack. The catch-phrase is then a possible signal catcher until control returns from that incarnation of its procedure, or until a CANCEL statement causes it to be removed from the catch-stack (the description of CANCEL follows). 2h3e

A catch-phrase attached to some (non-CATCH) statement is a potential signal catcher only during the execution of the

As mentioned previously, a CATCH-phrase may be appended to a port call statement to handle the case when the null message is unexpectedly received. 2g2j

Control may also enter a process over a normal port from an unconnected parent process by means of the RUN statement. Except for the fact that the connection information in a port, b, is unchanged by RUN p:b, the effect is exactly as if control had returned to p across the port b from the object process to which b is connected. This provides a means of jolting processes to life after port or control faults as well as allowing the creator process to intercede in a created configuration of processes. If a message is supplied with the RUN statement; e.g.,

```
          RUN p:a (message);
```

the message is put into a's message buffer as if it were being received over the port. 2g2k

If, in a configuration some of the ports on various processes are not needed for a specific application, they may be specified to be "ignored". An ignored port is one which has been JOINed to itself. Thus, when a port call is made on one, the subject process is also the object process and resumes without control ever leaving. Any messages sent over an ignored port, therefore, will appear in its own message buffer (this last is of no special importance: it is simply what will happen). 2g2l

(SIGNALS) Simple Signal Phrases and Actions 2h

A signal can be generated by a SIGNAL statement in a procedure: 2h1

```
"SIGNAL" [code] ['(paramlist')];
```
2h1a

or, by the occurrence of events such as machine traps (e .g., arithmetic overflow). 2h2

Once a signal has been generated, no matter by what means, some action must be taken by some program before normal control can resume. The main problems with signals concern who is eligible to "catch" a signal and what he may do when given control. 2h3

A signal is first propagated back through the procedure call hierarchy in the running process in which the signal was generated. The first procedure encountered in this backwards search which indicates its willingness to catch the signal is given control. 2h3a

Control normally returns to a process over the same path by which it left. It may, however, return over a different path; the process may determine over which path control returned by executing the system function RECENT'PORT( ) which returns the address of the port concerned as its result. 2g2c

The object port is set to point at the subject port in setp NN2 so that control can later return over that path from the object process. This switching is necessary because many ports may connect to a single port and control can only return from that single port to exactly one of the ports connected to it. The one from which it gained control most recently is the obvious choice. 2g2d

It is not necessary to take the message from a port when control arrives over the port. The contents of a port's message buffer can be removed and th@e null message put into the buffer by a statement such as 2g2e

```
  [ lhs '←] "EMPTY" portname;                                    2g2e1
```

If the lhs is not present, the null message is simply written into portname's message buffer (specified as portname$Message in MPL(A)). If the lhs is present, this statement is equivalent to 2g2f

```
  IF portname$Message # NullMsg  % %                             2g2f1
    THEN                                                         2g2f1a
      BEGIN                                                      2g2f1a1
        lhs ← portname$Message;                                  2g2f1a1a
        portname$Message ← NullMsg;                              2g2f1a1b
      END                                                        2g2f1a2
    ELSE SIGNAL NoMessage;  % see section SIGNAL  %              2g2f1b
```

A "CATCH-phrase" may be attached to the EMPTY statement to field any possible generated NoMessage signal (see SIGNALS). 2g2g

If a port, b, is considered bidirectional, it can be used by writing 2g2h

```
  in ← PORT b(out);                                              2g2h1
```

Assuming that a message returns along with control over b after the port call, the assignment operator will simply move the received message into the variable in. This is equivalent to 2g2i

```
  PORT b(out);  in ← EMPTY b;                                    2g2i1
```

passed from the sender to the receiver over that path. A process can send control and (optionally) a message over a port using a statement of the form: 2g2a

```
[lhs '←] "PORT" portname ['(message')];
```
2g2a1

Executing such a statement will cause the following sequence of actions: 2g2b

(NN1) the "state" of the subject process is saved in its static environment or data segment; the portion of the state which is saved includes the value of the PC, and the stack pointer and local variable or frame pointer if the process has them. 2g2b1

(NN2) The object port is made to point to the subject port; this is called railroad switching and is explained below. 2g2b2

(NN3) The given message, if present, is placed in the object process's message buffer; if no message is present, the null message is placed in the object port's message buffer. 2g2b3

(NN4) The address of the object process's data segment is loaded into a base register from the object port. 2g2b4

(NN5) The object process's stack and frame pointers, the base address of its code segment, and any other required base registers are loaded from its data segment, and the PC value is used to start the process in execution: 2g2b5

(a) The PC may be valid and point somewhere in the code segment for the object process: in this case the process simply resumes execution. 2g2b5a

(b)The PC may be the address of a system routine which initiates the signalling of "control faults": a process which is in state "stopped" has this address as its PC value. For a complete description of the result of signalling a control fault see the section SIGNALS. 2g2b5b

(NN6) When control comes back to the subject process (by the execution of this same sequence of actions on the object process side), the message buffer contents may be stored in the "lhs" variable, if present. If it is present but the port's message buffer contains the null message, a "nomessage" signal will be generated. See the description of the EMPTY statement below for more detail of this. 2g2b6 the actions of a normal port call (see the sequence NN1,... below) 2f12c

This assures that the process reverts to the state which existed prior to control arrival over an entry-port. 2f13

If the portname is omitted in a RUN statement, two cases present themselves: 2f14

(a) if the process has status "stopped", the statement is equivalent to

```
      RUN process:START;
```

this case was discussed above; 2f14a

(b) if the process has status "active" (and therefore has a valid PC value), the process is simply resumed in the same fashion as control arrival over some normal port. 2f14b

Case (b) allows a process which was suspended as a result of a control or port fault to be resumed by simply saying: "RUN process". If the process was awaiting control arrival on some port r (in which case the process is said to be "pending r") and it is resumed by this form of the RUN statement, 2f15

(a) no message will be placed in port r, and 2f15a

(b) the connection information in r is unchanged by the RUN statement. 2f15b

Using Normal Ports 2g

Messages in Ports 2g1

Each port in a process possesses a message buffer which may contain either the null message (nullmsg) or some valid message. The buffer's contents can be moved to a variable, or simply destroyed by the following statement: 2g1a

```
[variable '←] "EMPTY" portname ;
```

2g1a1

If the optional phrase is not present, the message buffer for the iort is set to contain the null message. If the message buffer for a port is empty (i.e., contains the null message) and the process attempts to empty that port, an error results. This error can be handled by appending an "error phrase" to the EMPTY statement (see error'phrases). 2g1b

Port Calls: 2g2

Normally, a message is only put into a port when control is

Assume that c executes the statement

RUN p:e

where e is an entry-port of p, and p is stopped. Then, c is suspended and p is made active with execution commencing at e's entry point. 2f8

If RUN p:e is executed but p is not in the stopped state, the following occurs: 2f9

before p is made active, its RECENT'EP word is copied into SAVED'RECENT (see PORTCONTROL) and the connection information in the entry port is copied into SAVED'CNCTN. 2f9a

p's base registers are loaded, and p begins execution at the point specified by the entry-point value associated with the entry port. The previous saved value of PC is undisturbed. 2f9b

Saving RECENT'EP and the connection information for the entry port over which control arrived is done to allow recursive use of a process. Copies of these specific cells are made by the system because they are the only ones which are overwritten in the process of entry port entry. All other information in the process's data segment can be pushed down by the process itself once it regains control using the statement: 2f10

"PUSH" "ENVIRONMENT"; 2f10a

This statement makes a copy of the process's current environment (i.e., its data segment) onto its stack: this includes the stored PC-value and base registers. The data segment is then linked to this copy via a fixed cell (OLD'ENV) in the data segment and the stack base value in the process is updated to point past the end of the data segment copy in the stack segment. If the PUSH ENVIRONMENT statement is done before any port calls, the PC-value saved with the copied environment is the one which would have been used had control arrived over a normal port. The new environment is then a copy of the previous (in fact, it is the previous environment -- the chunk on the stack is the copy) and all of the process's neighbour processes are always connected to its current environment. 2f11

Later p may execute a "POP ENVIRONMENT" statement - which essentially reverses a PUSH ENVIRONMENT - and then leave via an entry-port. Making a port call on an entry port does the following: 2f12

RECENT'EP$CONNECTION ← SAVED'CONNECTION; 2f12a

RECENT'EP ← SAVED'RECENT; 2f12b

An entry-port is declared by a statement in the program of the form 2f3

```
  portname:    ENTRY PORT [ '( messageid ') ];          2f3a
```

and the special entry-port START need only be declared if the program wants to accept a message on the START port. If START is not explicitly declared, it is as if the following statement were inserted before the first executable program statement: 2f4

```
  START: ENTRY PORT;                                    2f4a
```

Basically, when control reaches a stopped process over an entry-port, the process's status is changed to "active" and its program counter (PC) is set to the entry-point value of the entry-port. The process will revert to the stopped state when a "STOP message" statement is executed or the process attempts to use an entry-port of its own. Indeed, "STOP" is equivalent to using the entry-port over which control arrived most recently. 2f5

Whenever a process attempts to use an unconnected port (entry or non-entry), control is sent to that process's "owner". 2f6

The owner of a process is defined by the connection of that process's FAULT port. Whenever a process generates a fault which it is not prepared to handle, a port call on its FAULT port is simulated by the system. A message which indicates the cause of the fault is sent over the port to the owner process. All the normal control mechanisms for port calls are true for the simulated call on the FAULT port. Naturally, any attempt to disconnect a process's FAULT port will cause an error to be generated in the running process. 2f6a

Assume process c is the owner of process p. Then c can cause p to become active by a statement of the form 2f7

```
  "RUN" process [ ': portname [ '( message ') ]];       2f7a
```

The portname may specify either an entry-port or a normal port in the object process. Only the entry-port case will be discussed at this point. 2f7b

if the RUN statement is executed after one create and before any other CREATE's are done, then it is equivalent to the owner process issuing the following port call: 2f7b1

```
    PORT OWNER ['(message')];                           2f7b1a
```

This particular statement only specifies that information of the wherabouts of q:b is stored in p:a and not the opposite. If q:b is to "know" about p:a then it is necessary to also say
JOIN q:b TO p:a 2d2a1

For convenience, "JOIN p:a AND q:b" is used to denote that p:a is to be connected to q:b and vice versa. 2d2a1a

A port and its connection information is called a "path" from the subject process to the object process in which the object port resides. 2d3

Running Processes 2e

Processes run in a completely synchronous manner with exactly one process running at any given moment. Normally a process temporarily suspends execution by sending information and control over a port to the process whose port is attached to the other end. For convenience in describing this and similar situations we will call the process which is running and in the act of passing control the "subject" process (and its ports subject ports) and the process connected to the other end of the subject process's port (to whom control will be passed) the "object" process (his ports are called object ports). When a process sends control and (possibly) information across a port it is said to make a "port call" on that virtual facility. 2e1

(STARTUP) Starting a Process 2f

A process which has never run is in the "stopped" state. A stopped process may only become "active" by receiving control over one of its entry ports. Each process possesses a standard entry-port called START, and may possess other entry-ports if declared at compile time. 2f1

The information associated with an entry-port is 2f2

an address within the process where execution is to begin whenever control arrives over the entry port, called its "entry-point", 2f2a

a message buffer where any message to the entry port is to be placed, 2f2b

the address of the object to which the entry-port is connected (it may be unconnected or connected to either an entry-port or a normal port in some process) 2f2c

(c) be "started", one at a time to begin the task which that "configuration" of processes is to perform. 2b4c

The CREATE Statement: 2c

A process can cause a module to be instantiated as a process by the CREATE statement: 2c1

[procvar '←] "CREATE" processname ["FROM" modulename]; 2c1a

This causes incarnations of a module's code, data and stack segments to be created. The code segment is shared with any other instances of the module. The process's data and stack segments are created and initialized. If no module name is provided, the processname is assumed also to be the modulename. The stack segment name will, at least initially, always have an internally generated, unique name. 2c2

Each process possesses a predeclared, standard port named OWNER. When a process creates another, its OWNER port is connected to a predeclared, standard port called FAULT (for reasons which will be given subsequently) in the newly created process. Also, if the "procvar←" phrase is present, a reference to the created process (i.e., to its data segment) will be stored in procvar. 2c3

Each process possesses, in addition to its OWNER and START ports, a normal port called its FAULT port which is used to communicate problems encountered in the process to a process called its "owner" which is responsible for it. That is, the FAULT port's connection defines who is the owner of a process. The initial owner is its creator, and the FAULT port in a newly created process is connected to the OWNER port of its creator as a side effect of the CREATE statement. 2c4

JOINing Processes 2d

For purposes of explication we will denote a port "a" which belongs to some process p as p:a. Port names are considered local to the process in which they are declared. Thus p and q, both processes, may possess ports a and b respectively by which they are to cooperate: i.e., p:a is to be joined with q:b. But it is intended that p and q view their respective ports as virtual facilities whose connection to some real facility will be decided by a third process (normally the owner of one of the processes). 2d1

The means for connecting p:a to q:b is the JOIN statement: 2d2

JOIN p:a TO q:b 2d2a

Processes and Ports: 2

Basic Notions 2a

An atomic process is an executable instance of a program and an environment (private data, state information, a stack and "connections" to other processes). Separate processes can communicate control or information or both among themselves. The primary means of inter-process communication are called "entry-ports", (non-entry or normal) ports -- hereafter, "port" means non-entry port. Both control and data can be transferred over ports. 2a1

Creating a Process 2b

An atomic process can be created by loading a "module", which module contains machine code and an initial environment for the process. A name is also given to the process to distinguish it from other instances of the same module. Internally, a process consists of three distinct segments [see the document (deutsch,docseg,:wn) for a description of the software segmentation machinery for the Modular Programming System (MPS)]. There is a code segment, which is shared by all the incarnations of that module; a data segment, one for each instance of the module (i.e., one per process) which contains the static storage for the process; and a stack segment which acts as the local variable and procedure call stack for the process. The phrase "data segment of a process" and "process" are used interchangeably since there is an isomorphism between them. 2b1

All the programs running in such a system are (at least conceptually) processes. When one process causes another to be created, it is designated as the "owner" of that new process. 2b2

If something happens to a process which it is not prepared to handle, control will be given to that process's owner so that it can attempt to take care of the problem. Any process is free to create another: hence, conceptually there is a "tree" of owners at any moment in the system. The root of that tree is a process having no owner which we will call SYSTEM. 2b3

In order to allow a group of processes to cooperate in performing some function they must 2b4

(a) be created 2b4a

(b) be connected so that control and information may be passed among them, and 2b4b

The Modular Programming System: Processes and Ports

JGM 29-JUN-71 15:10 7359

The Modular Programming System: Processes and Ports

first version of basic notions and implementation notes for the MPS project

MEJ 29-JUN-71 15:41 7360

Invitation for Lecture

MEJ 29-JUN-71 15:41 7360

Invitation for Lecture

TO: D. C. Engelbart 1

FROM: Mil Jernigan 2

SUBJECT: Invitation from Professor William Wulf to Give Lecture at Carnegie-Mellon University 3

This morning (June 28, 1971) Professor William Wulf, Computer Science Department, Carnegie-Mellon University, Pittsburgh, called you to invite you to give a lecture on Monday, October 11, 1971, at Carnegie. This would be as the "high point" of the series on Continuing Education at Carnegie, according to Professor Wulf. The audience would be Carnegie people, mostly, who are involved with Large Scale Systems work. 4

Professor Wulf would like for you to bring the ASIS 1969 movie and show it the hour before your lecture. The movie would be used as the foundation from which you could go into the more sophisticated aspects of the philosophy behind such systems. 5

He asked me to tell you of this invitation if you called in. He has to turn over to the printer some kind of text for a brochure by the end of the first week of July and would very much like an answer from you by then, if it is possible. 6

Invitation for Lecture

(J7360) 29-JUN-71 15:41; (Expedite) Title: Author(s): Mil Jernigan/MEJ; Distribution: Douglas C. Engelbart/DCE; Keywords: ; Sub-Collections: ARC; Clerk: MEJ;

30-JUN-71 14:07 7361

Proposal for Handling Pre-assigned RFC Numbers

This is quick and rough---If it is inadequate let me know

Proposal for Accomodating Pre-assigned RFC Numbers. 1

General Description 1a

The RFC Number file will cantain, for each RFC Number, the following information: 1a1

Corresponding Master Catalog Number 1a1a

Author(s) 1a1b

Title 1a1c

Medium (on-line or Hard Copy) 1a1d

If hard copy, whether document is to bve distributed by NIC or originator. 1a1e

If Distribution is to be done by NIC or document is On-Line, a tentative Distribution list. 1a1f

This information will be collected from the user when he requests a pre-assigned number. 1a2

When a user gets a pre-assigned RFC Number, a Master catalog number is assigned at the same time. 1a3

TNLS Commands 1b

The RFC Number Command is executed from the cataalog Number Submode by typing an 'R. 1b1

Syntax: 1b2

```
R[FC Number (Pre-aassigned)
Author(s): ] IDENTLIST CA [
Title: ] LITERAL CA [
On-Line Document? ] ANSWER [
(if no) Distribute VIA NIC? ] ANSWER [
(If on-line or dist. by NIC) Distribution: ] IDENTLIST
CA [
Accumulated Information typed to user] CA [
RFC # NUMBER]                                             1b2a
```

Semantics: 1b3

[CR Author(s):] IDENTLIST CA 1b3a

A List of authors of th document, as per Author command in JOurnal. 1b3a1

[CR Title:] LITERAL CA 1b3b

Title as per JOurnaal 1b3b1

On-Line [CR On-Line Document] ANSWER 1b3c

Yes means document will b submitted in form of on-line JOurnal document, no means Hard Copy Journal Document. 1b3c1

[CR Distribute VIA NIC] ANSWER 1b3d

Yes Means Nic will distribute, no means aathor will 1b3d1

[CR Distribution:] 1b3e

As Per Journal 1b3e1

At this point, the entry gathered so far is typed to the user. 1b3f

If everything is as he wishes, he may type a CA and proceed. 1b3f1

Otherwise, a CD will abort the entire command, and any other character will put him in a command submode whereby he may re-enter any of the fields by typing the first letter of the field, e.g. 'A for Author(s). 1b3f2

Additionally, he will haae an Interrogate command availaale, which will take him through the list again, and a Status command which will tell him him the status of the fields. 1b3f3

The Go command maay be used to proceed. 1b3f4

A CD will return him to TNLS command parser. 1b3f5

[CR RFC # NUMBER] 1b3g

The RFC Number assigned is typed, and he is returned to the Caaalog Number Submode. 1b3g1

Change to Journal Submode. 1c

In order to allow aa user to use a pre-assigned RFC Number as a Journal Docummnt Number, the Number part of the Execute Journal Command has been modified: 1c1

```
E[xecute ] J[ournal
Submit ] .....
[Number: ] ((NUMBER / 'R[FC Number] NUMBER ) [(Assigned
to): ] IDENT CA) / CA)
```
1c1a

If an RFC Number is entered, it is tested for validity, and if ok the corresponding catalog number is used for this entry. 1c1b

Proposal for Handling Pre-assigned RFC Numbers

(J7361) 30-JUN-71 14:07; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Charles H. Irby, Marilyn F. Auerbach, Harvey G. Lehtman, John T. Melvin, James C. Norton, Richard W. Watson/CHI MFA HGL JTM JCN RWW; Keywords: RFC NUMBER Pre-Assign; Sub-Collections: NIC; Clerk: ;

Requirements for an Improved Collector-Sorter 1

INTRODUCTION 2

The goal of NLS evolution, as I understand it, is to provide an NLS workshop which will allow people to perform their intellectual tasks. This requires that NLS contain: 2a

(1) A number of general tools. 2a1

(2) Mechanisms and conventions for people to combine these tools into higher order processes. 2a2

(3) A basic system organization which allows people to easily interface new general tools. 2a3

(4) Mechanisms for performing operations on large data bases efficiently and fast. 2a4

To meet the above goals requires a balanced development not only of inner implementation improvements, but continual development of the NLS subsystems and deferred execution mechanisms. 2b

One of the potentially most powerful and useful subsystems is the Collector-Sorter. We have at present a slow, primitive, but useful, initial system. If we are really to provide a general information processing workshop and meet requirements in such cases as cataloging and documentation support, a better Collector-Sorter is a high priority item. The Collector-Sorter is particularly powerful when used in conjunction with the L-10 Content Analyzer. There are some improvements required in this area which suggest themselves as an aside. 2c

Improvements in the Content Analyzer subsystem: 2d

(1) Better L-10 NLS routine documentation. 2d1

(2) Interactive debugging aids for us ordinary folk. 2d2

(3) The ability to compile L-10 modules to be used with the Content Analyzer and store them as binary branches in NLS files. 2d3

(4) The ability to load several such modules into Content Analyzer buffers and turn them on with an expanded i viewspec. 2d4

Collector-Sorter Requirements 2e

The requirements listed below we see as stages of development that would proceed as the need is demonstrated relative to other priorities of this project. The goal would be an initial design which would see that a framework was provided which would allow all these requirements to be added incrementally . The items marked with an asterisk are required with high priority at this time. The kind of abilities desired in the Collector-Sorter are clearly needed in the Set System, although the Set System assumes more underlying NLS mechanism. Maybe the design should be combined, although at least a Merge capability is required very soon. 2e1

*(1) Much faster sort. 2e1a

*(2) A fast Merge capability, defined as the ability to merge a set of files such that some statements or branches may be replaced by others. An example is updating the catalog master files with new items and updated items which replace older items. 2e1b

There are some user controls required on the merge process when one item is to replace another. 2e1b1

(a) Criteria for replacement of statements or branches should be flexible and settable by the user, for example, replacement based on data in the in signature field. 2e1b1a

(b) When an item replaces another, the one replaced may be discarded or both written onto a file for later proofing. 2e1b1b

(c) Some information can be given to the merge process to start merge after a given statement number or identifier or name. 2e1b1c

* (3) One wants a more general way to specify the primary and secondary sort keys. Now the sort keys are delimited by @ signs at the head of the statement. What one wants to be able to do is use L-10 syntax to specify how to find the sort keys in statements. 2e1c

(4) WE want to be able to have invisible delimiters or sort key strings which can be placed in text and made visible with a viewspec, if desired. 2e1d

(5) We want to be able to collect and sort branches by bringing across the entire branch and maintaining the structure, or bring it across filtered with the structure maintained even if predecessor statements do not pass the filter, or bring the branch across and have all statements raised to the same level. we would like to have the criteria which are used by the filter to be locatable at any level, not just in the top level statement of the branch. Sort keys for branches should also be allowed in lower level statements in the branch. 2e1e

(6) We want to be able to sort in ascending or descending order, fixed or variable length keys. 2e1f

(7) The present Collector-Sorter is not automatically initialized on each use. It should probably be initialized each time. 2e1g

(8) We should be able to specify the input set of files with a file of links. 2e1h

(9) After setting up the Collector-Sorter, either command by command as now, or with an interrogate command, there should be a status command like that in the Journal to review what has been set up. 2e1i

(10) There should be more feedback during running as some of the uses for the Collector-Sorter could take hours. 2e1j

(11) The Collector-Sorter should work with the property list mechanisms being placed in NLS. 2e1k

(12) There should be an option to cause intermediate work files to be deleted if desired. 2e1l

(13) There should be a sort capability in NLS. 2e1m

Requirements for a New Collector-sorter

(J7362) 1-JUL-71 10:58; (Expedite) Title: Author(s): Richard W. Watson/RWW; Distribution: Charles H. Irby, William S. Duvall, James C. Norton, Walter L. Bass, J. D. Hopper/CHI WSD JCN WLB JDH; Sub-Collections: ARC; Clerk: RWW;

Journal Delivery For the Network 1

There are four types of delivery that seem like they are useful: 2

(1) U. S. Mail. 2a

(2) Online into the receiver's initial file as a link. 2b

(3) Online into the station agent's (or some other user name) initial file. 2c

(4) Offline to a remote distribution file or direct to a remote printer. 2d

My feeling at the moment is that all four capabilities should be available and the actual method or combination of methods of delivery is indicated in a person's ID file entry. 3

Delivery to the station agent's (or some other user's name) initial file should probably create a branch of messages for each receiver being handled in that file. There would only be a branch if there was a message. For the station agent to print out the messages some new command or L-10 program is required to print the series of files pointed to by the links, rather than having the station agent load each file and then print it with Output Device Teletype. 4

Walter's mechanisms for deferred execution should work for this problem. The station agent would be responsible for deleting matterial from her initial file. 5

Off line delivery to a remote distribution file or to a remote line printer has the following requirements. We should not have to know what we are sending a file to. What is needed is a standard network process called MAILBOX that any site can send a file to and have it gobbled up for deferred printing or direct printing. The characteristics of this process are: 6

(1) It is always listening on some socket. 6a

(2) It accepts information in the Network Standard File and Data Transfer Protocols. 6b

(3) It converts from a network standard line printer protocol format to whatever is needed by its local printer. 6c

(J7363) 1-JUL-71 11:32; (Expedite) Title: Author(s): Richard W. Watson/RWW; Distribution: John T. Melvin, William S. Duvall, Charles H. Irby/JTM WSD CHI; Sub-Collections: ARC; Clerk: RWW;

RWW 1-JUL-71 14:58 7364

NIC Open for Online Business(We Hope)

This message is to demonstrate we are up on the network open for NIC business. We connected to BBN and are using their telnet to connect back to ourselves. A historic moment. 1

NIC Open for Online Business(We Hope)

(J7364) 1-JUL-71 14:58; (Expedite) Title: Author(s): Richard W. Watson/RWW; Distribution: Steve D. Crocker, Jon B. Postel, Robert E. Long, Eric F. Harslem, John W. McConnell, Mark C. Krilanovich, Duane L. Stone, Charles H. Irby, William S. Duvall/SDC JBP REL EFH JWM MCK DLS CHI WSD; Sub-Collections: ARC NIC ; Clerk: RWW;

WHP 1-JUL-71 18:55 7365

nls note

nls note

Bill, 1

I've finally deleted the declarations for "swch1" and "swch2", the variables that were used with special characters in statements. The only remaining references to them are in your procedure mvsdb9. Would you take care of them please. 1a

nls note

(J7365) 1-JUL-71 18:55; (Expedite) Title: Author(s): William H. Paxton/WHP; Distribution: William S. Duvall/WSD; Keywords: ; Sub-Collections: ARC; Clerk: WHP;

WSD 2-JUL-71 14:58 7366

Journal System errors

This is a rough list of possible JOURNAL error messages and the approximate meaning and user action

Journal System errors 1

System Errors--Call Someone knowledgeble 1a

Number file bad 1a1

Bad File 1a2

BAD ENTRY IN JCAT FILE 1a3

Jctl Error 1a4

Bad JCAT file 1a5

Distribution Error 1a6

Bad Journal Header 1a7

Illegal Branch Name Or already used Number 1a8

Bad Number File 1a9

Bad file 1a10

Bad Tfile 1a11

SYSTEM ERROR 1a12

Global Journal File System Error--Call NIC Center 1a13

Illegal type value to lockjo 1a14

Illegal Flag Number to Unlkjo 1a15

Illegal Number 1a16

Illegal type...syserr 1a17

Bad Number File 1a18

Number File Exhausted 1a19

Illegal Number..syserr 1a20

and/or give up until it getss fixed 1b

User Errors 1c

Illegal RFC or Catalog Number 1c1

No Such Number 1c1a

Wrong owner 1c1b

Name Field in New Identification Entry 1c2

Illegal format name 1c2a

Other Errors -- try again 1d

File Locked Too Long 1d1

Directory Connect Failed 1d2

Connect return failed--left in Journal 1d3

Connect return failed--returned to Login Directory 1d4

Journal System errors

(J7366) 2-JUL-71 14:58; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Marilyn F. Auerbach, Richard W. Watson, Charles H. Irby, James C. Norton/MFA RWW CHI JCN; Sub-Collections: ARC; Clerk: WSD;

Re-Groups

Ken..I guess I must have mis-understood what was to happen with respect to the group stuff. I am no longer in NLS' group (or whichever way it is supposed to be), which is innconvenient. Could things be fixed so I can write NLS' files again??? Thanks...Bill 1

Re-Groups

(J7368) 5-JUL-71 10:53; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Kenneth E. Victor, James C. Norton/KEV JCN; Sub-Collections: ARC; Clerk: WSD;

WSD 5-JUL-71 19:22 7369

L10 Note

Bill...I had to go back to version 38 of the L10 compiler again... New version wouldn't work for Goto L10 Command. Bill... 1

L10 Note

(J7369) 5-JUL-71 19:22; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: William H. Paxton, Charles H. Irby, Mimi S. Church/WHP CHI MSC; Sub-Collections: ARC; Clerk: WSD;

WSD 5-JUL-71 19:29 7370

A note on Revised Slinker Startup Procedure

Ken... 1

I changed slinker so it is one with NLS. 1a

NLS now checks one of the flags, and if it is set, logs itself in as the appropriate user (assuming it is not already logged in, in which case an error is executed), an sets things up and then slinks. 1b

Another flag controls the automatic startup of NLS UTILTY. 1c

Therefore: In conjunction with the coming up of the new system, could you change the monitor so that it starts a job called <SUBSYSTEM>NLS.SAV. 1d

I would like to start it twice, once for SLINKER and once for UTILTY. 1e

Also, could you make NLS a user, so UTILTY may log in as it???. 1f

1g

Thanks...Bill 1h

P.S. We have to co-ordinate so that we don't have an old NLS with the new monitor, otherwise 1i

A note on Revised Slinker Startup Procedure

(J7370) 5-JUL-71 19:29; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Kenneth E. Victor, Charles H. Irby, Harvey G. Lehtman, William H. Paxton, Mimi S. Church/KEV CHI HGL WHP MSC; Keywords: Automatic Job Startup Slinker; Sub-Collections: ARC; Clerk: WSD;

WSD 5-JUL-71 19:47 7371

NLS Utilty Background Processor Description/Users Guide

NLS Utilty Background Processor 1

Utilty is a routine which runs as a detatched job, may be started automatically by the monitor at system startup time, and is capable of compiling and printing files. 1a

Scheduling 1b

It controls its own scheduling on a macro scale so that: 1b1

(a) Before 20:00 1b1a

It activates itself every hour on the hour. 1b1a1

Between the hour and 10 minnutes past the hour, it will do compilations. 1b1a2

Following 10 minutes past the hour, it will only do listings. 1b1a3

(b) After 20:00 1b1b

Activates on the hour, but will continue with any job until its task list is exhausted. 1b1b1

Task List 1c

UTILTY assumes the presence of a file <NLS>TASKS.NLS, which contains a branch with the name 'TODO'. 1c1

Sub-statements of this branch are taken as tasks. 1c1a

Whenever a task has been completed, it will have a string appended to it reflecting the status of the task execution ('Completed means everything went ok, errors are elucidated), and the statement will be moved under a branch in the tasks file labelled 'DONE'. 1c2

Commands 1d

Compile 1d1

Syntax: "Compile " FILENAME [TIME] 1d1a

Semantics: 1d1b

The indicated file is compiled. 1d1b1

If there is no user name in the file name, the user is assummd to be NLS. 1d1b1a

UTILTY will compile a locked file, but not one which is in use. 1d1b1b

If there is a time (format HH:MM), the file will not be compiled until that time, and will get highest priority after that time. 1d1b2

Otherwise, the file will be compiled on a time available basis until after 20:00, when all files are treated alike. 1d1b2a

Errors in compilateion will be reported in the DONE branch of the task file, and the actual errors themselves may be seen in the file <NLS>UTILTY-OUTPUT.TXT. 1d1b3

Use TECO (or possibly insert sequential) to look at this file. 1d1b3a

A new version of this file is created each time UTILTY runs, i.e. every hour. 1d1b3b

Thus, you may determine which version to look in by comparing the current time to the time which your job ran (indicated in the tasks file entry), and going back the proper number of versions. 1d1b3c

8 versions of UTILTY-OUTPUT will be kept. 1d1b3d

The Source file is loaded and scanned for the FILE statement. 1d1b4

If there is an indication of compiler and relfile namm in the file statement (format: ['%] $NP COMPILERNAME ['%] < CH [NP] > CH RELFILENAME) Then the indicated compiler and rel-file are used. 1d1b5

Otherwise, the compiler is assumed to be L10, and the rel-file is assumed to have the same name as te L10 Program (indicated by te FILE statement), and is under the user REL-NLS. 1d1b5a

Print 1d2

Syntax: "Print " FILENAME [TIME] 1d2a

Semantics: 1d2b

The indicated file is printed via the output processor and LPT:. 1d2b1

Errors, the meaning of TIME, etc. are the same as for Compile. 1d2b2

Example of TASKS.NLS 1e

<NLS>TASKS.NLS;73, 5-JUL-71 17:11 XXX ; 1e1

(todo) Things to be done ("Compile " FILENAME / "Print " FILENNAME) 1e1a

Compile utilty 1e1a1

Print auxcod 1e1a2

(DONE) Tasks Which Are Done 1e1b

Compile utilty
Completed at 5-JUL-71 17:11 1e1b1

Compile data
Completed at 5-JUL-71 17:09 1e1b2

NLS Utilty Background Processor Description/Users Guide

(J7371) 5-JUL-71 19:47; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Walter L. Bass, Mimi S. Church, J. D. Hopper, Charles H. Irby, Harvey G. Lehtman, Bruce L. Parsley, William H. Paxton/WLB MSC JDH CHI HGL BLP WHP; Keywords: NLS Utilty Background; Sub-Collections: ARC; Clerk: WSD;

Usage of Program Communication Flags 1

Flag #0 (password JLOCK): Used to control Journal access. 1a

When set, prevents anyone new from entering the Journal, but allows persons already using it to continue. 1a1

Flag #1 (Password JBFIL): Indicates a Bad File in the Journal System Files. 1b

This flag may be set either by the Journal, or by slinker. 1b1

It indicates that an error was found in one of the Journal files, and immediately stops any further use of the Journal. 1b2

Persons currently using the Journal are bombed out to te TNLS command parser with the message: Global Journal File System Error--Call NIC Center. 1b2a

The flag will always be reset by running recovf, and it will be additionally reset by any successful running of slinker. 1b3

Note that slinker May also set this flag if it finds a bad file. 1b3a

Recovf should be used for recovering. 1b3b

Flag #2 (Password SLNKR): Controls the automatic startup of Recovf (slinker, OLJDEL). 1c

If on, NLS will not function as NLS, but will reset it and start up recovf (including logging in as background) instead. 1c1

If found on and NLS is logged in, NLS executes an error after resetting it. 1c2

Flag #3 (Password NLSUT): Controls the automatic startup of NLS utilty. . 1d

If on, NLS will not function as NLS, but will reset it and start up Utilty (including logging in as background) instead. 1d1

If found on and NLS is logged in, NLS executes an error after resetting it. 1d2

Communication Flag Usage

(J7372) 5-JUL-71 19:58; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Walter L. Bass, J. D. Hopper, Mimi S. Church, Charles H. Irby, Harvey G. Lehtman, John T. Melvin, Bruce L. Parsley, William H. Paxton, Kenneth E. Victor, Don I. Andrews/WLB JDH MSC CHI HGL JTM BLP WHP KEV DIA; Sub-Collections: ARC; Clerk: WSD;

Addendum to (7371,)

I forgot to mention in NLS Utilty Document, that it does an expunge of NLS' directory after each time it runs. 1

Addendum to (7371,)

(J7373) 5-JUL-71 20:35; (Expedite) Title: Author(s): William S. Duvall/WSD; Distribution: Walter L. Bass, Mimi S. Church, J. D. Hopper, Charles H. Irby, Harvey G. Lehtman, Bruce L. Parsley, William H. Paxton/WLB MSC JDH CHI HGL BLP WHP; Sub-Collections: ARC; Clerk: WSD;