* * * DRAFT * * *

DDN TELNET/HP3000

Internal Design

Information Networks Division

Location Code: 66-7850

Project Number: 6651-3008

July 24, 1986

Katy Jenkins

PRODUCT IDENTIFICATION

SECTION 1

Name	HP3000 DDN Services: TELNET
Mnemonic	none
Project Number	6651-3008
Project Manager	Bruce Templeton
Project Engineer	Katy Jenkins
Product Manager	Dennis King
Product Assurance	Murali Subbarao
Documentation	Mike Genevro
Off-Line Support	Bruna Byrne
On-Line Support	Susan Gennrich Lorraine Mehrtens Jim Zepp
Software Production	George Melchiorsen

DESIGN OVERVIEW

SECTION 2

TELNET, a network virtual terminal protocol, is one of three required services protocols defined by the Department of Defense for the Defense Data Network (DDN). The other services are File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP).

The DDN architecture model has also defined lower layer protocols, including Transmission Control Protocol (TCP), Internet Protocol (IP), and X.25. The existing HP implementations of these layers require modifications in order to be certified with the Defense Data Network. An overview of the product requirements necessary to provide DDN compatibility for HP3000s is given in DDN Services/HP3000: Investigation Report.

This document discusses the design of the TELNET protocol modules. It is recommended that the External Specifications be read before this document. Familiarity with MIL-STD-1782 (the formal definition of the TELNET protocol) would also be useful.

2.1 DESIGN APPROACH

The following are some of the general guidelines used in the design of the TELNET protocol modules. They appear in no particular order.

- Design with the stated project priorities in mind. Specifically, schedule is the highest priority for this project, followed by functionality, followed by performance.
- Provide at least the minimum functionality necessary to insure certification with the DDN.
- Leverage from existing code whenever possible.
- Provide a product which is independent from any particular release of the MPE operating system.
- Design the product with portability to other systems in mind.
- Facilitate foreign language customization of the product.
- Provide a clean, maintainable, understandable product.

2.1.1 Project Priorities

A timely, functional, easy to use product at first release is an important design goal of the TELNET project. However, since a goal of timeliness often conflicts with goals of functionality and usability, design tradeoffs must be made to achieve an acceptable combination of all three. One example of such a design tradeoff is the way in which an HP3000 terminal user invokes system break on a remote system over a TELNET connection. (See the External Specifications for more details.) To invoke system break on a remote system while running the TELNET user program in data mode, the terminal user must:

1) Enter the TELNET attention character to switch from command mode to data mode.
2) Type the TELNET user program SEND BREAK command.
3) Type REMOTE to return to data mode in order to receive the remote system's response.

Although this interface is somewhat awkward for the terminal user, this approach was considered preferable for first release, given the complexity of implementing a more user-friendly approach (i.e. allowing the terminal user to invoke system break on the remote system simply by striking the (BREAK) key on the local terminal).

Likewise, the product must have an acceptable performance level at first release and performance considerations will be incorporated throughout the design and implementation process. The major concern for first release is to provide a functional product with reasonable performance which can be optimized and enhanced in subsequent versions.

2.1.2 Certification

The design of the TELNET/3000 product is adhering to the TELNET protocol specification contained in MIL-STD-1782 to insure compliance with the DDN requirements. Additionally, the Defense Communication Agency (DCA) has published a draft, Defense Data Network Host Interface, Qualification Testing: Higher Level Protocols, discussing test procedures for certifying with the DDN. The progress of this document is being tracked, and the TELNET/3000 implementation is designed to pass the certification tests currently outlined in the qualification testing document.

2.1.3 Code Leverage

Communication between HP machines is currently provided by the Network Services (NS) product. The Virtual Terminal Service (VT) portion of the NS product provides terminal users the capability to logon to remote systems and to run applications as if they were using a terminal directly attached to the remote system. The TELNET protocol provides a similar type of terminal to host communication facility between various types of HP and non-HP systems which provide the TELNET service.

The NS/VT and TELNET services, however, differ greatly in the protocol used to exchange information between the peer protocol entities communicating across the network. These protocol differences limit the amount of leverage which can be obtained from the NS/VT code.

The TELNET design will use the NS/VT architecture model, and the system interfaces of the TELNET modules will be similar to those of VT. (By similar I mean that the types of interfaces will be the same, i.e. a pseudo terminal driver which intercepts MPE I/O requests, an application monitor process which is created by DSDAD, etc. The information exchanged across the interfaces, i.e. which MPE I/O requests are supported, contents of port messages sent from pseudo terminal driver to application monitor process, etc. will be different for TELNET.) Use of the NS/VT architecture model will allow TELNET to use many of the common services which were developed for NS, such as the buffer management routines and the DSDAD control process. The TELNET module system interfaces will be discussed in more detail later in this document.

There will be little direct code leverage from the internal workings of the VT modules, due to the differences between the two protocols and their traffic characteristics. Experience gained from working with the NS/VT code, however, will be very valuable in the design and implementation of the TELNET modules.

2.1.4 MPE Independence

The TELNET design will not have any dependencies on a particular version of the MPE operating system (other than a limitation that the operating system be an MPE V/E version). One design implication of this goal is that the TELNET user program is activated with an MPE :RUN command, as opposed to a specific MPE command interpreter command like NS/VT. The use of the :RUN command also simplifies the interfaces for the TELNET module on the local terminal user system.

2.1.5 Portability

The TELNET program modules (the terminal user routines and most of the application server routines) will be written in PASCAL to facilitate porting of code to non-HP3000 machines. Application server routines which access system tables and all of the pseudo terminal driver code will be written in SPL.

A few observations should be noted regarding the portability of the TELNET/3000 code. Most HP3000 virtual terminal products by necessity are intimately associated with the MPE operating system and its terminal I/O characteristics. Thus it only makes sense to talk about portability to systems which resemble MPE (such as MPE/XL systems). Likewise, the usefulness of porting the TELNET/3000 code depends upon the availability of associated products (such as the NS common services) on the target system.

2.1.6 Foreign Languages

The TELNET modules will load all command keywords, error messages, help messages, etc. from a message catalog. This will allow localization by a Network Manager.

2.1.7 Maintainability

The TELNET modules will be designed to provide a clean, easily understood, product which is maintainable by a non-author. Code comments and supporting documents (ES and IMS) will be kept up to date and the code implementation will follow standard structured programming practices.

2.2 OVERVIEW OF OPERATION

This section provides a guide to the design and operation of TELNET/3000. It is assumed that the reader is familiar with NS Services, NS transport, MPE ports, and Network Manager; if not HP internal documents exist to aid the reader.

2.2.1 TELNET Local Structure

Figure 1 shows the structure of the TELNET/3000 product on the terminal user's local system.

2.2.1.1 Startup

A terminal user in an MPE session on the HP3000 asks the CI to run the TELNET user program, thereby creating the TELNET user process as a child of the user's session.

The TELNET user program will open the MPE session input and output devices (typically the user's terminal) in order to communicate with the terminal user.

The TELNET user process will check the Port Dictionary to determine if the TELNET service has been allowed. The "TELNETL" entry in the dictionary will be placed there by DSDAD when the Network Manager includes the service in an NSCONTROL START command. If the port entry is not found, the TELNET user program will print an error message on the user's terminal and terminate.

When the terminal user requests a connection to a remote host (by specifying a nodename info string in the :RUN command or by issuing a TELNET user program OPEN command), the TELNET user program will open a NetIPC connection to the remote host.

2.2.1.2 Steady State

The TELNET user program will communicate with the user's terminal using file system intrinsics and ATTACHIO calls. The TELNET user program will communicate with the network by using NetIPC intrinsics.

While in data mode, the TELNET user program will process data read from the user's terminal and send it out on the NetIPC connection. The TELNET user program will process data received from the network and write it to the user's terminal.

In command mode, the TELNET user program will suppress output of any data received from the network and will interpret data read from the user's terminal as TELNET user program commands to be executed.

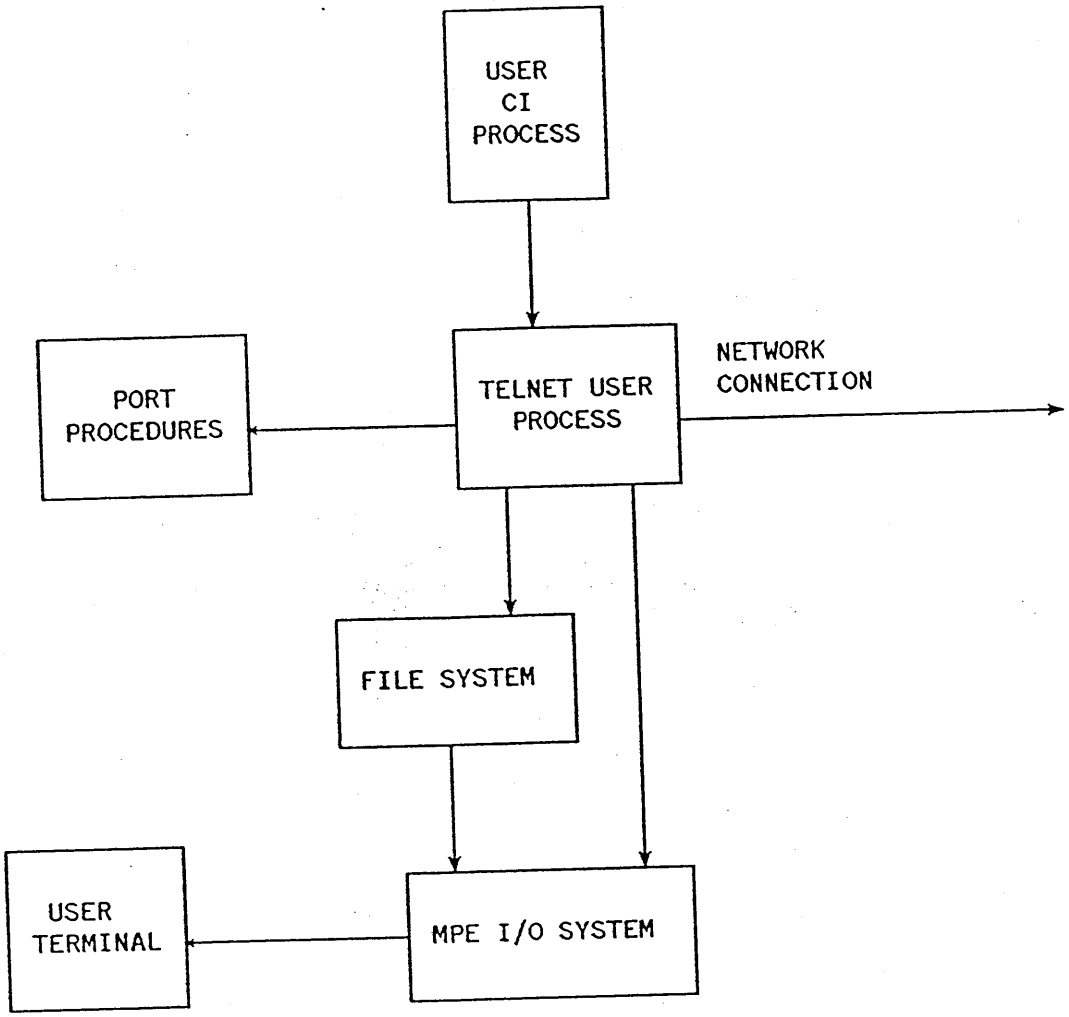

Figure 1 - TELNET Local Structure

2.2.1.3 Shutdown

The NETIPC connection will be closed when the terminal user enters the TELNET user program CLOSE or EXIT commands, or opens a new connection with the OPEN command (and responds affirmatively to the "Close current connection?" prompt). The connection may also be closed if an irrecoverable NetIPC error occurs. The TELNET user process remains alive until an EXIT command is entered, or the father CI process is aborted.

2.2.2 TELNET Remote Structure

Figure 2 shows the structure of the TELNET/3000 product on the remote system.

2.2.2.1 Startup

If the TELNET service has been enabled via an NSCONTROL START command, DSDAD will listen on Port 23 for a connection request. When an incoming connection request is received, DSDAD will create a TELNET server process and give the NetIPC connection to this newly created process. The TELNET server process creates an MPE port to receive further communications from DSDAD and from the TELNET pseudo terminal driver. The TELNET server process is a child of DSDAD at this point.

The TELNET server process will obtain and initialize a TELNET pseudo terminal. The MPE device recognition sequence will be started for the TELNET pseudo terminal. This should cause MPE to issue a colon prompt to the pseudo terminal.

Any I/O requests (for example, the colon prompt) directed to the pseudo terminal will be received by the TELNET pseudo terminal driver. The pseudo terminal driver will notify the TELNET server process of work to be done by sending an MPE port message to the TELNET server process. The TELNET server process will send the colon prompt to the terminal user's local system using NetIPC intrinsics.

Data received from the network will be used to satisfy read requests which are issued for the pseudo terminal. No real work can be done however, until the terminal user types in "HELLO user.acct" to initiate an MPE session on the remote system. Once an MPE session has been established on the remote system, the TELNET server will adopt into that session.

2.2.2.2 Steady State

The TELNET pseudo terminal driver receives I/O requests directed to the TELNET pseudo terminal. The pseudo terminal driver cooperates with the TELNET server process associated with this pseudo terminal in order to handle the application I/O requests. The TELNET pseudo terminal driver informs the TELNET server process of work to be done by sending an MPE port message to the server. The TELNET server informs the TELNET pseudo terminal driver of work to be done by PCALing the driver. Data written to the pseudo terminal is sent out on the network connection by the TELNET server process. Data received on the network connection by the TELNET server process is used to satisfy read requests issued to the pseudo terminal. Additional details on the interaction between the TELNET server process and the TELNET pseudo terminal driver are provided in the "Module Design" section of this document.

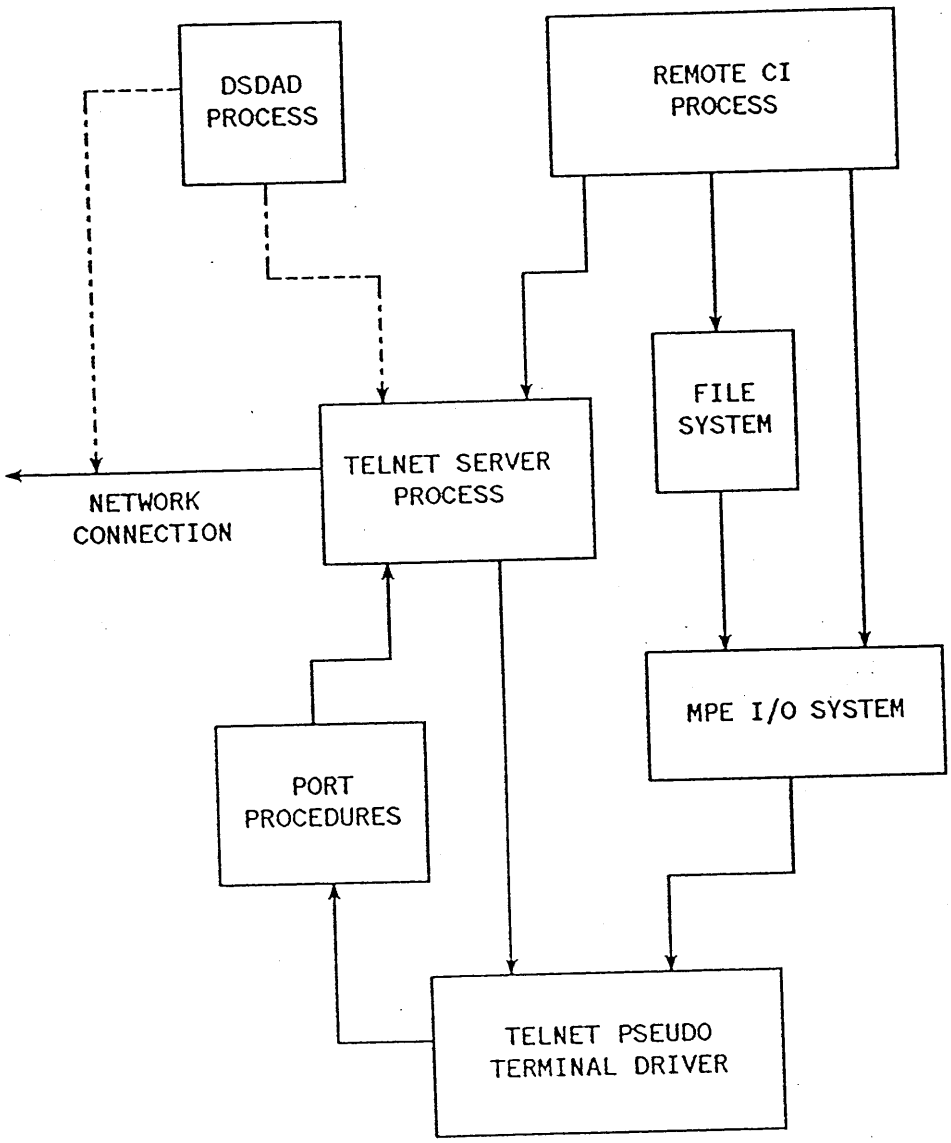

Figure 2 - TELNET Remote Structure

2.2.2.3 Shutdown

When an MPE session associated with the TELNET pseudo terminal is terminated (i.e. the terminal user typed :BYE, or typed :HELLO to relogon, or the session is aborted), the TELNET pseudo terminal driver will notify the TELNET server so that the server can adopt itself back under DSDAD. Any file information associated with this session will be reset. If the terminal user is not doing a relogon, the TELNET pseudo terminal will be deallocated and the NETIPC connection will be closed. If the terminal user is relogging on, the pseudo terminal will remain allocated, the NETIPC connection will be retained, and the TELNET server will adopt into the new MPE session once it is established.

If an irrecoverable error occurs on the NetIPC connection the TELNET server process will adopt itself back under DSDAD, abort the associated user MPE session, and cleanup and deallocate the associated TELNET pseudo terminal.

Whenever the TELNET server closes the NetIPC connection, processing is basically finished from the server's point of view. The TELNET server process will send a "Server Done" port message to DSDAD. Whether the TELNET server process is actually terminated or just put into reserve is determined by DSDAD, and is dependent on whether pcreated servers are being used.

NOTE

The decision to terminate the NetIPC connection when the user is not relogging on is based on observation of existing TELNET implementations. The TELNET protocol standard itself does not address the issue of shutdown. All of the TELNET server implementations which I have seen terminate the network connection when a user logs off. If the TELNET/3000 server retained the NetIPC connection when a user logged off, a problem could be created for non-HP3000 TELNET local systems which do not implement a TELNET attention character. The terminal user would not have a method to wake the local TELNET system out of data mode. Having the TELNET server close the network connection upon user logoff accomplishes this wakeup purpose. A terminal user who wishes to establish a new MPE session without terminating the NetIPC connection may do so by relogging on (i.e. typing :HELLO user.acct while currently in an MPE session).

2.3 MAJOR MODULES

The TELNET product will consist of three major modules, the user program module, the server process module, and the pseudo terminal driver module. The user program module and the server process module will be contained in one program file. The pseudo terminal driver is contained in a program file which is configured as an MPE I/O driver. Additionally there are a few utility routines which must reside in the system SL. These are the routines which provide the null terminal interface to FTP and must reside in the system SL so that FTP can call them. These SL routines could either reside in a separate SL segment set aside for DDN services or they could be incorporated into an existing SL segment which will always be present when DDN services are present (such as the ASUTIL segment).

2.4 MAJOR DATA STRUCTURES

The TELNET modules will use NS buffer management intrinsics to buffer data in the process stacks of the TELNET user program and the TELNET server process. No extra data segments will be used. The user program and server process will maintain global variable information in their process stack. The pseudo terminal driver will maintain global information in the MPE Device Information Table (DIT) associated with the pseudo terminal. The pseudo terminal driver may also access global information and the NS buffers which are located in the server process stack. The TELNET modules will need to access MPE system tables associated with the pseudo terminal, such as the Logical-Physical Device Table (LPDT) and IOQ elements. MPE Include Files will be used in the code to provide the definitions of these tables. Other MPE tables may be accessed indirectly as a result of routines the TELNET modules call (such as setting up Port Dictionary entries, or changing the father of the server process). The TELNET modules will access the DDN catalog file and will create a trace file in the user's local group and account if tracing is enabled.

2.5 MAJOR INTERFACES

Figures 3 and 4 summarize the major interfaces of the TELNET modules.

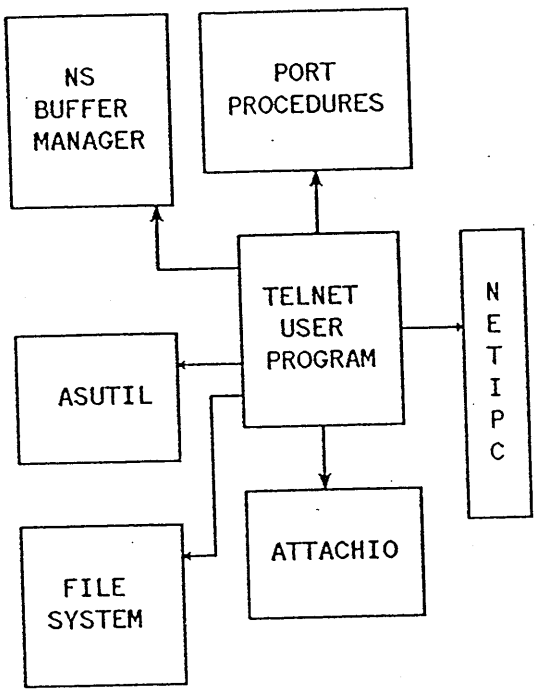

Figure 3 - Local TELNET Module Interfaces

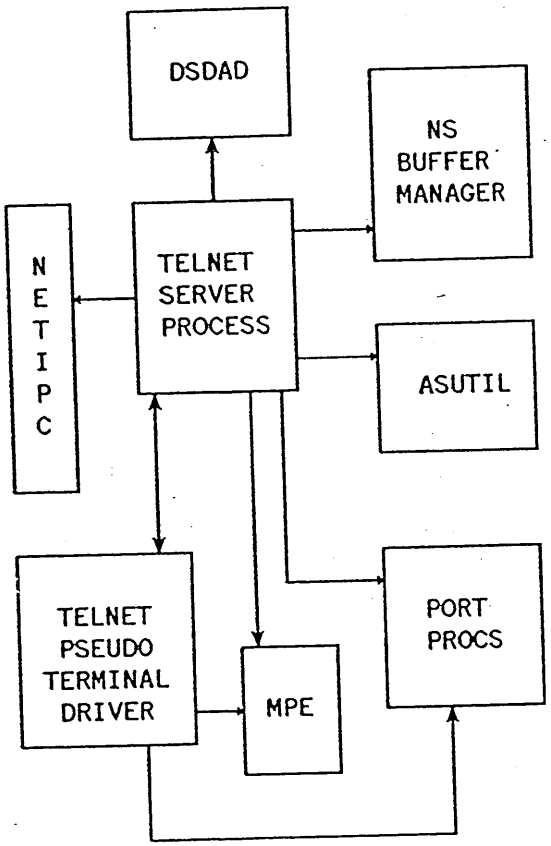

Figure 4 - Remote TELNET Module Interfaces

1) **Buffer Management**: The NS buffer management intrinsics will be used to manage buffers in the user program and server process stack. No extra data segments will be necessary. The NS buffers will be designated as "frozen" and "in use" since the TELNET modules need to use the actual address of these buffers in most cases. The interface is described in NS/3000 Overview Internal Maintenance Specifications.
2) **NetIPC**: All circuit connections, controls, sends, receives, and closures will be done through NetIPC intrinsics which are defined in NS/3000 User/Programmer Reference Manual. All communication with lower layers will be done through NetIPC intrinsics.
3) **ASUTIL**: The TELNET user program and server process will use NS/3000 utility routines which are contained in the ASUTIL segment. Examples of these utility routines are error logging routines and process termination cleanup routines. The interface to these routines is described in NS/3000 Overview Internal Maintenance Specifications.
4) **Port Procedures**: The TELNET user program will do a port dictionary lookup to determine if the TELNET service has been allowed. (On the server side this lookup is performed by DSDAD.) The TELNET user program will create a port which may be used occasionally as part of the TELNET

attention character processing. The TELNET server process will create a port for communication with DSDAD and the TELNET pseudo driver. The interface to the port procedures is described in Port Procedures: External Specifications.

5) **DSDAD**: The TELNET server process will communicate with DSDAD using the standard port messages outlined in NS/3000 Overview Internal Maintenance Specifications. A new port message(s) may be defined for DDN Services to enable server tracing.
6) **File System**: The TELNET user program will use file system intrinsics to communicate with the user's terminal. This interface is described in MPE Intrinsics Reference Manual.
7) **Attachio**: The TELNET user program will occasionally call the MPE procedure ATTACHIO directly in order to communicate with the user's terminal. Direct calls to ATTACHIO will be used only when the desired functionality is not available through the file system interface. The ATTACHIO interface is described in the code comments for the procedure ATTACHIO in the MPE module HARDRES.
8) **MPE**: The TELNET server process and pseudo terminal driver will access MPE tables related to the pseudo terminal. The format of these tables is given in MPE V Tables Manual for MPE V/E.

2.6 PERFORMANCE CONSIDERATIONS

The primary design technique employed by the TELNET modules to aid performance is the use of data concatenation. On the server system, the TELNET modules will buffer information from multiple application I/O requests. This information will be concatenated to reduce the number of NetIPC sends performed. Concatenated information may include application write data and/or TELNET commands.

On the user program side, data received from the network will be written to the terminal as it is received.

Terminal read data will normally be sent out on the network when an end-of-line condition (the user enters (RETURN) or the TELNET push character) occurs in LINE input mode. The primary intention in CHAR input mode will be to expedite the sending of terminal read characters and no explicit wait until the end of line will be done.

The interaction between the TELNET server process and the pseudo terminal driver has been designed to aid performance by minimizing deliberate process switches between the two modules while still avoiding possible deadlock situations.

2.7 LOCALIZATION CONSIDERATIONS

DDN Services will include a message catalog with the product. All TELNET keywords, delimiters, help messages, and error messages will be accessed from this catalog. This mechanism will allow one to change these items in the file and to use the MAKECAT program to make it a valid catalog file. Any errors occurring in the use of file system intrinsics will be reported by using the FERRMSG intrinsic to retrieve the appropriate file system error message from the system message catalog.

MODULE DESIGN

SECTION 3

3.1 TELNET PROGRAM

3.1.1 Description and Function

This section describes the outer block of the TELNET program. This program file is used by both the user process and server process, so one of the first things the program does is determine whether it is a user or a server and proceed accordingly. Some initialization is common for both user and server processes, (such as turning off arithmetic traps, etc.) so this initialization is done first.

3.1.2 Interface Specifications

If the program is running as a user process, the info string may contain an optional nodename entered by the user. If the program is running as a server process, the info string contains the DSDAD port id.

3.1.3 Algorithm

The basic algorithm of the program outer block is:

```
Initialize;  {do common initialization}
CheckFather; {determine if user or server process}
If UserProcess
  then UserProcessing
  else ServerProcessing;
```

3.2 USER PROCESSING

3.2.1 Description and Function

This procedure is the main body of the TELNET user program. It does user program specific initialization and then enters an infinite loop. This loop will be exited when the terminal user enters an "EXIT" command, and the program will be terminated.

3.2.2 Algorithm

The basic algorithm of the user processing is:

```
UserInit;
While True Do
  Begin
  CommandModeProc;
  DataModeProc;
  End;
```

In the discussion which follows, the Command Mode procedures will be referred to collectively as the CM and the Data Mode procedures will be referred to as the DM.

3.2.3 Local Data Structures

There are two types of data structures used by the User Processing routines: state variables and data buffers. Two sets of state variables are maintained: connection state variables, and terminal state variables.

Connection state variables record the state of the TELNET connection (open or closed, what TELNET options are enabled, etc.). This information is used by the CM routines to determine what commands are allowed (i.e. a limited set of commands is allowed before a connection is open). The connection state variables affect the actions taken by the DM routines (such as disabling local terminal echo if the connection is in TELNET remote echo mode). The connection state variables may be updated by either the CM routines (in response to a user command such as OPEN) or by the DM routines in response to TELNET negotiations received from the network.

Terminal state variables record the state of the user's terminal environment when in TELNET data mode. This information determines the behavior of the DM routines. Examples of terminal state variables are input, editing, and output modes. Terminal state variables are updated by both the CM and DM routines.

The user may obtain information about some of the state variables through the SHOW command.

3.2.4 Submodule Descriptions

This section describes the three major submodules of the User Processing procedure: Command Mode procedures (CM), Data Mode procedures (DM), and the Subsystem Break Trap Procedure (TRAP).

3.2.4.1 Command Mode Procedures

The CM routines repeatedly read and execute TELNET user program commands input by the terminal user. While in command mode, terminal reads and writes are done with waited I/O and the normal HP3000 terminal editing characteristics apply.

Execution of a command is completed before the terminal user is prompted for another command. SEND commands will not actually transmit information to the remote system. Information from SEND commands will be stored into the network buffers used by the DM. This information will be transmitted by the DM routines after the terminal user enters a REMOTE command.

If the terminal user enters a SEND command and the network buffers are full, the CM routines will temporarily exit to allow the DM routines to empty the network buffers. However, the user is still logically considered to be in Command Mode and the DM routines will not perform Data Mode terminal input and output but will simply return to the CM routines when room is available in the network buffers. This mode switching will be transparent to the terminal user (although a message indicating that the SEND may take a little longer to complete this time could be output if desired).

While the terminal user is in Command Mode, the TRAP procedure for the TELNET attention character is disabled. It will be reenabled when the terminal user returns to Data Mode by entering a REMOTE command. A user who wishes to terminate the TELNET user program while a command is executing in Command Mode may do so by breaking and aborting the program. The OPEN and SEND commands (when the network buffers are full) are the only commands which might take a noticeable amount of time to execute.

Command Mode will be exited when the terminal user enters a REMOTE or EXIT command. If a REMOTE command is entered, the user will be switched to Data Mode and the DM routines invoked. If an EXIT command is entered, resource cleanup will be performed and the process will terminate.

3.2.4.2 Data Mode Procedures

The DM routines continually perform two main tasks. Terminal input data is read, formatted according to TELNET protocol conventions, and sent out on the network. Data received from the network is processed and written to the user's terminal.

Writing data to the terminal takes precedence over reading. Data Mode terminal reads will be issued only if there is nothing to write. If write data arrives while a terminal read is pending, the read will be aborted if the terminal user is using TELNET NOWAIT Output Mode. Otherwise, the data will be written as soon as the terminal read completes. When using TELNET NOWAIT Output Mode, a quiesce IOQ will be issued when switching from writing to reading to prevent a loss of write data if the read should subsequently be aborted. (The quiesce IOQ will work on some, but not all terminal drivers).

Data Mode terminal writes are done with waited I/O. Data Mode terminal reads are performed with nowait I/O, as are IPCSENDs and IPCRECVs. Normally, one IPCSEND will be outstanding at a time. A second IPCSEND may be performed for the user SEND CLEAR command which uses TCP urgent data mode. The DM routines will concatenate terminal input data while an IPCSEND is pending. However, any available data will be sent as soon as the previous IPCSEND completes (i.e. the concatenation will not

wait for a maximum data length or timeout). The amount of available data sent will depend partially on the terminal user's choice of LINE or CHAR Input Mode. The choice of Input Mode does not explicitly affect the concatenation; rather it determines the maximum length specified in terminal reads and thus affects the amount of data returned when these reads complete.

The DM routines will periodically check to see if a subsystem break has occurred. If one has occurred, processing will be stopped and the DM routines will exit to allow the CM routines to run.

The DM routines will exit to Command Mode if an irrecoverable error occurs on the connection. They will also return to Command Mode when network buffer space becomes available if they were invoked by the CM for this reason.

3.2.4.3 Subsystem Break Trap Procedure

The TRAP routine is called by the MPE operating system if the TELNET attention character is input while the terminal user is in Data Mode. This routine simply sets a global flag in the user process stack and exits. The subsystem break causes any pending terminal I/O to complete successfully as soon as the trap procedure exits. This wakes up the TELNET user process and gives it a chance to check the global flag set by TRAP.

A potential hole exists with this method if the TELNET user process calls IOWAIT with no terminal I/O pending. This would occur only if the pipeline backed up and the DM network send buffers were full. In this case, the TELNET user process would not issue a data mode terminal read because there would be no place to put processed terminal input data. If a subsystem break occurred in this situation, the TELNET user process would not wake up until an IPCSEND or IPCRECV completed.

This delay in invoking command mode would be frustrating to a terminal user who wants to issue a SEND CLEAR command to clear up the pipeline. What I propose in this situation is to have the TRAP procedure send a port message to the TELNET user process to complete the IOWAIT. The TRAP routine would check flags set by the TELNET user process and only send a port message in this special instance. Multiple port messages would not be sent since the trap procedure, once executed, is disabled until explicitly reenabled by the TELNET user program.

3.3 SERVER PROCESSING

3.3.1 Description and Function

This procedure is the main body of the TELNET server process created by DSDAD. It allocates and initializes a TELNET pseudo terminal. The TELNET server process and the pseudo terminal driver cooperate to handle application I/O requests directed to the TELNET pseudo terminal. This section will discuss both of these modules. The TELNET server process routines will be referred to as the SP, the pseudo terminal driver routines will be referred to as the PD.

3.3.2 Algorithm

The PD may be thought of as an MPE IOQ manager; the SP is a NetIPC connection manager. All NetIPC sends and receives are performed by the SP. There is no shared connection as in NS/VT. Likewise, all explicit IOQ manipulation is done by the PD. The SP does not need to know exactly what an IOQ looks like.

The SP and PD communicate in the following fashion. As part of initialization, the SP creates an IOWAIT port to receive port messages from the PD (and DSDAD). The SP also allocates two buffers for application I/O request information. Information about the port and buffers is passed to the PD as part of its initialization. Logical ownership of the buffers is passed between the SP and the PD. Only the current owner is allowed to access the buffer.

The PD is the initial owner of the outgoing request buffer. The PD concatenates information about I/O requests requiring SP action into this buffer. This information is in the form of request "blocks" containing relevant information copied from the IOQ. When the PD wants to pass ownership of the buffer to the SP (so the SP can start processing), the PD sends a port message to the SP indicating this fact. The PD then considers the buffer to be "owned" by the SP and will defer processing of any further I/O requests which require SP attention until the SP has returned the buffer.

When the SP is finished with the outgoing request buffer, it returns it by PCALing the PD. Status information for the PCAL may be exchanged by the two modules through a local array set up in the SP stack for this purpose.

The SP is the initial owner of the incoming data buffer. This buffer contains terminal user input data which has been received from the network. Ownership of this buffer is passed through the PCAL mechanism. The SP PCALs the PD at appropriate times (discussed later) to give the PD a chance to review the data contained in this buffer and use it to complete application read requests. The PD is considered to own the buffer for the duration of the PCAL and will update the data descriptors accordingly if data is removed. Once the PCAL completes, the SP once again owns the buffer.

3.3.3 Local Data Structures

The PD and SP maintain separate data areas. The PD uses the terminal DIT as a global context area maintained between invocations. The SP maintains state variables in its process stack. The two modules do not mess with each other's data structures (except for a field in the pseudo terminal DIT which the SP will set to indicate that the SP is trying to invoke the PD). Information is passed via the mechanisms described above and each module updates its own data structures when information is received.

The PD variables are concerned primarily with the pseudo terminal state. The SP is concerned with the NetIPC connection. Some overlapping information is maintained by both modules. For example, the PD needs to know if a NetIPC connection exists and the SP needs to know the pseudo terminal echo state since this may affect TELNET option negotiations. The SP informs the PD of changes in the connection state by PCALing the PD with this information contained in the local array. The PD informs the SP of changes in the pseudo terminal echo state by formatting I/O request blocks into the outgoing request buffer and sending a port message to give the buffer to the SP.

3.3.4 Submodule Descriptions

This section provides additional information on the algorithms used by the PD and SP routines. It also includes comments on some utility routines which actually reside in a system SL segment.

3.3.4.1 Pseudo Terminal Driver Procedures

The PD outer shell is pretty much the same as any MPE terminal I/O driver. The PD while active, basically executes in a loop, handling any available I/O requests. The PD exits when there are no more I/O requests to process or when processing of I/O requests must be deferred until some PD external event occurs. The PD procedures are reentrant in the sense that multiple instances of the PD may be activated at any one time. However, each new instance of the PD immediately checks the DIT to see if another instance is active, and if so, exits immediately to avoid concurrent access to data structures. Any currently running instance of the PD therefore, must make sure that ALL currently available work is processed before exiting. The PD may be invoked by either the MPE I/O system or the SP. The PD will give priority to the SP requests. If a currently running instance of the PD is busy processing MPE I/O requests and notices that the SP is trying to call it, it will finish processing the current MPE I/O request and exit so that the PD instance created by the SP PCAL may execute. It is ok for the PD to leave some MPE I/O requests unprocessed in this situation, since the PD knows it will be immediately invoked by the SP. The PD does not perform any tasks which might cause it to block for an indefinite period of time, so the PD should be able to respond to SP PCAL invocations in a timely manner. The PD does not issue IPCSENDs and only one port message is outstanding at a time to the SP.

The PD can process and complete many types of application I/O requests on its own without help from the SP. Most of these are I/O requests resulting from FCONTROLs which simply update the pseudo terminal state. A few FCONTROLs (such as enabling/disabling terminal echo) require action by the SP to change the TELNET connection state.

In instances when an I/O request requires SP action, the PD can usually copy the relevant information into an I/O request block within the outgoing request buffer, update its own PD terminal state variables, and complete the request by returning the IOQ. This is how write IOQs are handled. The PD copies the application write data to the I/O request buffer, inserting any characters necessary for carriage control and prespacing or postspacing operations. The write IOQ is returned to the caller as soon as the data has been copied.

Two types of requests which the PD does not complete immediately are read requests and device close requests. When these requests arrive, the PD concatenates an I/O request block into the outgoing request buffer to inform the SP of their arrival. The PD must then wait for notification by the SP before completing these requests. The PD waits on a read request until the SP calls the PD to allow the PD to review the contents of the incoming data buffer for data to satisfy the read request. The PD waits on a device close to give the SP a chance to adopt out of the user session and back under DSDAD before the user session terminates. (The SP had adopted into the user session when the PD had informed it of a logon request arrival).

Of course, if the PD does not currently own the outgoing request buffer, it must wait for the SP to return it before processing new application I/O requests which require action by the SP.

The PD may concatenate information from multiple application I/O requests into the outgoing request buffer before giving ownership to the SP. The PD will stop concatenation and give control of the buffer to the SP when one of the following conditions occurs: 1) the outgoing request buffer is full so there is no room for new requests, 2) a concatenation timer expires (used to insure a reasonable response time), 3) the last application request processed was one which the PD cannot complete until it receives notification from the SP. The PD processes application I/O requests in order of their arrival (with exceptions for preemptive writes) and this order is maintained by the I/O request blocks in the concatenation buffer. Note that the PD may process and complete I/O requests which can be handled entirely by the PD without having to stop concatenation of I/O request blocks requiring SP attention.

3.3.4.2 Server Process Procedures

The SP is responsible for managing the NetIPC connection. The SP uses nowait IPCSENDs and nowait IPCRECVs to transmit and receive data from the network. The SP calls IOWAIT to complete any pending IPCSENDs or IPCRECVs and also to receive port messages from the PD or DSDAD.

When control of the outgoing request buffer is passed from the PD to the SP, the SP processes the information contained in the buffer. Application write data is copied into the SP network send buffers and application control request information is used to update SP state variables and possibly to generate TELNET command sequences to be placed into the network send buffers. The SP returns the outgoing request buffer to the PD as soon as it has processed all information contained within.

The SP processes terminal input data received from the network, and places it into the incoming data buffer. It then checks to see if it has received notification of a read request from the PD. If a read request is pending, the SP calls the PD to give it a chance to review the data. The PD will return a status of either read complete or read incomplete. If the read is incomplete (because the PD did not find a read termination condition in the terminal input data) the SP will call the PD the next time it adds more data to the terminal input buffer. If the read is complete (either the data satisfied the read request, or the read had been aborted), the SP will not call the PD again until it receives notification of a new read request. The PD is responsible for updating the buffer data descriptors if it uses data from the buffer to complete a read request. Note that if a read times out or is aborted, data in the incoming request buffer is not discarded; it is saved and used to satisfy the next read request. If the incoming data buffer becomes full (because the application is not issuing read requests) the SP will stop accepting data from the network. The SP will continue to listen to the network, however, by issuing IPCRECVs with the "preview" option. This allows the SP to find out about connection closings or urgent data pending. Urgent data is used as part of the TELNET signal to clear the data path, in which case the SP can discard the buffered terminal input data.

When a connection is going to be terminated because the associated session is logging off, the SP will insure that all send data has reached the partner TELNET entity before closing the connection.

3.3.4.3 Utility Procedures

The procedures used for the null terminal interface reside in a system SL segment so that they can be called by FTP. The null terminal procedures need to allocate a pseudo terminal and PCAL the pseudo terminal driver for initialization just like the SP does, so common utility routines to accomplish this task are also contained in the same SL segment. When a PD is initialized as a null terminal, there is no corresponding SP. A null terminal PD simply completes all I/O requests immediately with a successful status except for command interpreter read requests. The null terminal PD holds onto the CI read request as a means of suspending the CI.

DATA STRUCTURE DESIGN

SECTION 4

This section provides information about the input/output buffer requirements and usage for the TELNET modules. Optimal sizes for the buffers will be determined in the implementation and testing phases.

4.1 USER PROCESS BUFFERS

The TELNET user process will use the following buffers:

Command Input Buffer
Command Output Buffer
Terminal Input Buffer
Terminal Output Buffer
Network Send Buffers (2)
Network Receive Buffer

The command input buffer is relatively small, since only one command is read at a time. The command output buffer is also small, being used to output the command prompt, any command output, and error/help messages.

The terminal input data is used to receive input from data mode terminal reads. This buffer must be as large as the largest data mode terminal read. Data from this buffer is processed and put into the network send buffers. This buffer is non-concatenable, i.e., a new data mode terminal read is not issued until all data from a previous data mode terminal read has been processed.

The network send buffers contain data and TELNET commands which are ready to be transmitted onto the network. These buffers have two states: ReadyforData and DataSent. Two buffers are used, so that one buffer can be filled with processed data from the terminal input buffer while waiting for an IPCSEND of the other buffer to complete. These buffers are concatenable when in the ReadyForData state. Processed data from multiple data mode terminal reads may be concatenated into a ReadyForData buffer. Once an IPCSEND is issued for the buffer and it enters the DataSent state, the buffer will not be ReadyForData again until the IPCSEND completes.

The network receive buffer is used to receive data from the network. This data is processed and placed into the terminal output buffer. The network receive buffer is non-concatenable and data from the network will not be accepted until all previously received network data has been processed. IPCRECVs with the preview option may be issued to listen to the connection.

The terminal output buffer contains data which is ready to be written to the terminal. Since terminal writes are done with waited I/O, data in this buffer is normally not concatenated. However, processed data from several IPCRECVs may be concatenated if the user is using TELNET WAIT Output Mode and the data mode procedures are waiting for a data mode terminal read to complete before issuing a terminal write.

4.2 SERVER PROCESS BUFFERS

The TELNET server process will use the following buffers:

Outgoing Request Buffer
Incoming Data Buffer
Network Send Buffers (2)
Network Receive Buffer

The outgoing request buffer is used to receive concatenated application I/O request blocks from the pseudo terminal driver. This buffer must be as large as the largest single PD application I/O request block which depends on the largest terminal FWRITE allowed. Information from this buffer is processed and put into the network send buffers. This buffer is not returned to the PD until all information contained within has been processed.

The network send buffers contain data and TELNET commands which are ready to be transmitted onto the network. These buffers have two states: ReadyforData and DataSent. Two buffers are used, so that one buffer can be filled with processed information from the outgoing request buffer while waiting for an IPCSEND of the other buffer to complete. These buffers are concatenable when in the ReadyForData state, although sending of buffers will not be delayed simply to wait for more requests. The PD concatenation algorithm should insure that reasonably sized IPCSENDs are done. Once an IPCSEND is issued for a buffer and it enters the DataSent state, the buffer will not be ReadyForData again until the IPCSEND completes. Only one IPCSEND will be outstanding at a time.

The network receive buffer is used to receive data from the network. This data is processed and placed into the incoming data buffer. The network receive buffer is non-concatenable and data from the network will not be accepted until all previously received network data has been processed. IPCRECVs with the preview option may be issued to listen to the connection.

The incoming data buffer contains terminal input data which is ready to be edited by the PD and delivered to the application. Processed data from multiple IPCRECVs will be concatenated into this buffer while waiting for an application to read it.

INTERFACE DESIGN

SECTION 5

Most of the TELNET module interfaces have been discussed in detail either previously in this document, or in the External Specifications. The Internal Maintenance Specifications will contain details on the exact formats used in the I/O request blocks and the PCAL array to exchange information between the TELNET server process and the pseudo terminal driver. The IMS will also contain detailed information on the state variables and buffer descriptors maintained by each module.

5.1 SUPPORTABILITY DESIGN

This section discusses some of the design techniques which will be used to:

- Increase the supportability of the product.
- Increase the ability to get high PFA coverage for the TELNET modules.
- Aid in measuring performance bottlenecks in the TELNET modules.

The NetIPC tracing facility will be used by the TELNET modules to trace calls to NetIPC intrinsics and data sent and received over the network. The TELNET process modules (user and server) will maintain an internal logging buffer in the process stack to trace internal states and procedure calls and to maintain statistics. The pseudo terminal driver will not maintain a logging buffer, but will maintain flags in the terminal DIT to indicate what portion of code it is currently executing. The TELNET user and server processes will use the NS logging procedures (i.e. AS'LOG'ERROR) to log serious error conditions. Additionally, the TELNET user program will print error messages to the terminal user (in command mode).

A combination of DEBUG accessible variables and conditionally compiled code will be used to generate false error conditions in order to test error recovery mechanisms.

IMPLEMENTATION PLAN

SECTION 6

6.1 CODING CONVENTIONS

The TELNET user processing and server processing procedures will be written in PASCAL. MPE Intrinsics will be used for the user terminal I/O and control, with the exception that an ATTACHIO call will be made to issue a quiesce IOQ. The pseudo terminal driver and associated utility procedures in the system SL will be written in SPL. In-code comments will constitute 40-50% of the final finished source code.

6.2 IMPLEMENTATION ORDER

The following outline presents the planned implementation order for the TELNET modules.

1. Primary User Processing Module Routines
   a. Outer Block of TELNET Program
   b. Outer Block of User Process
   c. User Process Initialization / Data Structure Setup
   d. CommandMode Procedure Outer Block
   e. Command Input and Parsing Routines
   f. Individual Command Execution Routines
   g. Data Mode Procedure Outer Block/ Control Loop
   h. Data Mode Terminal Input/Output Routines
   i. Data Mode NetIPC Input/Output Routines
   j. Subsystem Break Trap Procedure Excluding Port Message Interface
   k. User TELNET Data Transformation Routines
   l. User TELNET Option Negotiation Routines

2. Primary Server Processing Module Routines
   a. Server Process Initialization / Data Structure Setup
   b. Server Process Outer Block / Control Loop
   c. Server Process NetIPC Input/Output Routines
   d. Server Process Outgoing Request/Incoming Data Buffer Management
   e. Server Process Utility Procedures/ Null Terminal Interface

3. Primary Pseudo Terminal Driver Routines
   a. Pseudo Terminal Driver Initialization / Data Structure Setup
   b. Pseudo Terminal Driver Outer Block / Control Loop
   c. Pseudo Terminal Driver Outgoing Request Buffer Management
   d. Pseudo Terminal Driver Individual I/O Request Handling
   e. Pseudo Terminal Driver Incoming Data Buffer Management
   f. Pseudo Terminal Driver Null Terminal Handling

4. Secondary Server Processing Routines
   a. Server Process Individual I/O Request Block Handling

b. Server TELNET Data Transformation Routines
c. Server TELNET Option Negotiation Routines

5. Secondary User Processing Routines
   a. User Process Port Message Handling Routines
   b. Subsystem Break Trap Procedure Port Message Interface
   c. User Process Help Facility and Output Formatting

Table of Contents

Section 1
PRODUCT IDENTIFICATION

Section 2
DESIGN OVERVIEW

Section 3
MODULE DESIGN

** END OF FORMATTING **
TDP/3000 (A.04.01) HP36578 Formatter
THU, JUL 24, 1986, 7:23 PM
NO ERRORS
INPUT = IDTELNET.TELNET.DDN
OUTPUT = *HP2680

```
                                                #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
                                                #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
                                                #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM

                                        #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
                                        #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
                                        #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM

                                #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
                                #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
                                #J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM

#J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
#J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
#J458; #O1861  *  IDJOB, KATY.DDN; SLP  *  THU, JUL 24, 1986,  7:45 PM
```