3.0 CPCI DESCRIPTIONS

This section presents requirements for the government specified physical interface and CPCIs FTP, SMTP, and DDN Gateway. It also presents the required NOS and DCNS version/level and all required modifications to NOS and DCNS, including X.25. Only the system interface requirements for the CPCIs IP, EGP, TCP, and TELNET are given in this section; full external and protocol specifications for these CPCIs are given in separate ERSs (refer to section 2).

3.1 Physical Interface

3.1.1 Physical Interface Abstract

The physical interface provides the actual electrical connectivity between the CYBER host, terminals, and the DDN.

3.1.2 Physical Interface Description

The physical interface consists of four distinct interfaces: the interface to a DDN PSN, the terminal interface, the inter-DI interface, and the CYBER interface. All physical interfaces are implemented using standard CDCNET hardware and software with no changes required for DDN. Refer to sections 3.11 and 3.12 for descriptions of the CDCNET hardware CI and software CPCI.

3.1.2.1 DDN PSN Interface (XHIF 2.4, XHIF App.B, IMP App.J)

The physical interface to a DDN PSN shall adhere to the requirements of the DDN X.25 Host Interface Specification (XHIF) and the Specification of the Interconnection Between a Host and an IMP (IMP). The physical interface to a DDN PSN shall be provided by the 2610-16 Line Interface Module and 2610-150 cable.

a. The protocol shall be RS-449 Balanced.

b. The connector type shall be DB37. The optional 9-pin connector is not used.

c. The operating mode shall be DTE, with DCE mode provided by the PSN or modem.

d. The DCE connection shall be either direct connection or via modem.

e. Line rates of both 56Kbps and 9.6 Kbps, in two directions simultaneously (full-duplex), shall be supported.

f. The LIM shall reside in the NDI.

3.1.2.2 Terminal Interface

The physical interface to terminals shall be provided by the 2612-1 Line Interface Module and 2612-550 cable.

a. The protocol shall be RS-232.

b. The connector type shall be DB25.

c. The operating mode shall be DCE, with DTE mode provided by the terminal.

d. Line rates of .3Kbps, 1.2Kbps, and 9.6Kbps, at a minimum, shall be supported.

e. Standard CDCNET terminal types shall be supported, including VT100, CDC-721, and basic Teletype.

f. The LIM shall reside in the TDI.

3.1.2.3 Inter-DI Interface

The physical interface for DI inter-connection shall be provided by the 2608-1 Ethernet Serial Channel Interface and 2608-110 transceiver cable.

a. The transceiver interconnections shall be provided by the Ethernet Transceiver and Ethernet cable, or by the 2631-1 Ethernet multiplexor.

b. The physical protocol shall be IEEE 802.3 CSMA/CD.

c. The link-level protocol shall be IEEE 802.2 Link.

d. The ESCI shall reside in the MDI, the NDI, and the TDI.

3.1.2.4 CYBER Interface

The physical interface to the CYBER shall be provided by the 2607-001 Mainframe Channel Interface and CYBER peripheral channel cable.

a. The MCI shall reside in the MDI.

3.2 X.25

3.2.1 X.25 Abstract

X.25 provides network-level access to the DDN PSN, and shall use RS-449 for the physical access.

3.2.2 X.25 Description

The X.25 protocol provides network access to the DDN PSN and consists of the X.25 link level and X.25 packet level protocols. The standard CDCNET HDLC, X.25 Packet Level, and X.25 Network Solution software shall be used, with

minor modifications required for compliance with DDN specifications. The X.25 protocols shall reside in the NDI. Figure 3.2-1 depicts the various X.25 components and their relationship to other DDN CYBER host interface and CDCNET components.

Since the size of modifications to X.25 is insufficient to justify a separate IDS document, the detailed design for these modifications is incorported into the ERS, in section 3.2.4.

3.2.2.1 X.25 Link Level Protocol (XHIF 1.2.1, 2.3)

The X.25 link level protocol shall be provided by the CDCNET High-level Data Link Control (HDLC) Stream Service Routine (SSR) and CIM controlware.

3.2.2.1.1 HDLC Protocol (XHIF 1.2.1, 2.3)

The protocol shall adhere to the requirements of the DDN X.25 Host Interface Specification (XHIF). No changes are required to CDCNET to support the DDN requirements for this protocol.

a. The HDLC procedure shall be Line Access Procedure/Balanced (LAPB).

b. The protocol shall support line rates of both 9.6Kbps and 56Kbps full-duplex.

3.2.2.2 X.25 Packet Level Protocol (XHIF 1.2.1, 2.1, 2.2)

The X.25 packet level protocol shall be provided by the CDCNET X.25 Packet Level module and the X.25 Network Solution module. The protocol shall adhere to the requirements of the DDN X.25 Host Interface Specification (XHIF). The supported service type shall be DDN Standard X.25.

Changes to CDCNET X.25 to support DDN fall into the following functional areas:

a. X.25 Network Solution support of the X.25 facility for DDN Standard service.

b. X.25 Network Solution support of the X.25 facility for DDN precedence.

c. X.25 Network Solution suppression of CDNA-specific data in the X.25 Call-User-Data on DDN circuits.

d. X.25 Network Solution suppression of CDNA-specific 3A header in X.25 data packets.

e. X.25 Packet Level support of the X.25 diagnostic packet.

The requirements for the above changes are given in the following subsections.

3.2.3 External Interfaces

The following subsections define the X.25 external interfaces for the DDN project. Since X.25 is a standard CDCNET product, the only interfaces specified are those that must be modified or those that affect system interface testing.

3.2.3.1 X.25 Network Configuration

The operational parameters for CDCNET X.25 are configured via operator commands. The following subsections specify the changes required to these command processors for DDN X.25. Additionally, values for existing parameters are specified where required by the design. Parameter values that are required by the DDN connection but are not required by the design are specified in section 3.2.8.1.

3.2.3.1.1 Precedence Parameter (XHIF A-3.2-4)

The Define_X25_Network command processor shall be extended to allow specification of DDN precedence (refer to ARH6785, Network Definition, sec 9.5.4).

a. A parameter shall be defined called DDN_X25_Precedence (alias dxp).

b. The permitted values for the dxp parameter shall be in the range 0 to 3.

3.2.3.1.2 Protocol ID Parameter (XHIF 2.1.3)

The Define_X25_Network command processor shall be used to specify the DDN protocol id. No changes to the standard CDCNET X.25 are required to support this parameter (refer to ARH6785, Network Definition, sec 9.5.4).

a. The network_protocol_id parameter shall be specified with a value of CC(16) for DDN Standard X.25 circuits.

b. The value of npi shall be used to identify DDN specific processing as defined in later subsections.

3.2.3.1.3 Architecture Type Parameter

The Define_X25_Network command shall be used to specify the supported network architecture. This parameter identifies the network architectures to be supported by the network solution, and can be cdna, dod, or both. No changes to the standard CDCNET X.25 are required to support this parameter (refer to ARH6785, Network Definition, sec 9.5.4).

(Note that this feature is present in CDCNET level 1704, but is not a documented or supported feature at this level. This does not affect DDN implementation, but may effect PSR support during test.)

a. The architecture_type parameter shall be specified with a value of 'dod' for DDN Standard X.25 circuits.

b. The value of at shall be used to identify DDN specific processing, as defined in later subsections.

3.2.3.1.4 PDN Name Parameter

The Define_X25_Interface command shall be used to specify the DDN pdn name. No changes to the standard CDCNET X.25 are required to support this parameter (refer to ARH6785, Network Definition, sec 9.5.5.2).

a. The public_data_network parameter shall be specified with a value of 'DDN_STANDARD' for DDN Standard X.25 circuits.

b. The value of pdn shall be used to identify DDN specific processing as defined in later subsections.

3.2.3.2 X.25 Peer Protocol (XHIF 1.2.1, 2.1, 2.2)

The X.25 modules shall conduct the peer protocol with a DDN packet switching node. The following subsections specify the changes required to the X.25 protocol implementation for DDN X.25.

3.2.3.2.1 X.25 Private Facility for DDN Standard (XHIF 2.1.2.1)

DDN X.25 defines a special private facility that must be specified if a virtual circuit is to utilize DDN Standard X.25 service. Standard service implies that the X.25 link is terminated at the PSN, and all subsequent routing is done using DDN Internet Protocol. Standard service is available only when IP is used for the 3B layer.

The X.25 Network Solution module shall be modified to support the DDN Standard call facility.

a. The call facility shall be included in the CALL REQUEST packet if and only if the associated interface was configured with pdn=DDN_STANDARD and the associated network was configured with npi=CC(16).

b. The call facility shall be two octets in length with the value 0401(16).

3.2.3.2.2 X.25 Private Facility for DDN Precedence (XHIF 2.1.2.2)

DDN X.25 defines a special private facility that allows a host to select a non-default precedence for the virtual circuit. This precedence value determines the priority given to one user circuit over another during heavy load conditions.

The X.25 Network Solution module shall be modified to support the DDN Precedence call facility.

a. The call facility shall be included in the Call Request and Call Accept packets if and only if the associated network was configured with the dxp parameter specified.

b. The call facility shall be two octets in length with the value 080n(16) where n is the value specified in the dxp parameter.

3.2.3.2.3 CDNA User Call Data (XHIF 2.1.3)

The current X.25 Network Solution server passes a CDNA-specific block in the user-call-data portion of the Call Request and Accept packets that is used by the remote network (assumed to be CDNA) to select the appropriate network id with which to associate the virtual circuit. DDN Standard X.25 neither sends this block, nor recognizes such a block when received. The block is not required for a DDN Standard X.25 network solution because the associated network id is always unambiguous.

The X.25 Network Solution module shall be modified to suppress CDNA-specific data for DDN Standard X.25 networks.

a. The CDNA Call User Data Block shall be sent with the Call REQUEST packet if and only if the associated interface was configured to support the cdna architecture type (at=cdna or at=(cdna,dod)).

b. The protocol id field of the Call User Data Block shall be sent even for DDN connections.

c. The CDNA Call User Data Block shall be fabricated for incoming Call Request or Call Confirm packets if and only if the associated interface was not configured to support the cdna architecture type (at=dod).

d. The CDNA Call User Data Block shall be fabricated with the following information (refer to ARH6211, CDCNET X.25 Interface, sec 5.7.2):

   1. The DI System ID shall be of the form 0800250nnnnn(16) where nnnnn are the 5 right-most hexadecimal digits of the assigned network id specified in the X25 network configuration command (refer to ARH6836, CDCNET Titles and Addresses, sec 8.2.2). This ID is used for consistency and identification, and should not be used by CDNA modules; the address must be regarded as a foreign network host. The number has been chosen such that it will not collide with other system addresses as long as the network id's assigned to DDN connections are no more than 5 digits.

   2. The Network ID shall be the network id specified in the X.25 network configuration command (refer to ARH6785, Network Definition, sec 9.5.4).

3.2.3.2.4 CDNA 3A Header

Intranet 3A currently appends a header to all data blocks sent over an X.25 network solution, and expects all data blocks received to contain the header. This header follows the same format as the IEEE 802.2 link level header (refer to IEEE standard 802.2-1985), and is used to identify the destination SAP (that is, protocol id of 3A ULP). DDN Standard X.25 neither sends this header, nor recognizes such a header when received. The header is not required for a DDN X.25 network solution because the associated SAP is always unambiguous - it is always the IP SAP/protocol id.

The X.25 Network Solution module shall be modified to suppress the 3A header on data packets destined for DDN Standard X.25 networks.

a. The CDNA 3A header shall be sent with a data packet if and only if the associated interface was configured to log the cdna architecture type (at=cdna or at=(cdna,dod)).

b. The CDNA 3A header shall be fabricated for incoming data packets if and only if the associated interface was not configured to support the cdna architecture type (at=dod).

c. The 3A headers shall be stripped or added only for the first packet of each X.25 complete message, since only those packets contain the 3A headers.

d. The CDNA 3A header shall be fabricated with the following information (refer to ARH6694, IEEE 802 Support and CDNA Layer 3a Headers).

    1. The Destination SAP field (DSAP) shall be set to the IEEE-assigned IP protocol id. The value of this protocol id is 96(10) (refer to RFC-948, Transmission of IP Datagrams over IEEE 802.3 Networks).

2. The Source SAP field (SSAP) shall be set to the same value as the DSAP field (refer to 1 above).

3. The control field shall be set to zero.

3.2.3.2.5 X.25 Diagnostic Packets (XHIF 2.2-6)

DDN X.25 sends diagnostic packets when certain errors are detected by the DCE. The DCE expects these packets to be accepted without response or action. CDCNET X.25 discards diagnostic packets without notice, so that an administrator will not know if such a packet is received.

The X.25 Packet Level module shall be modified to support diagnostic packets from the DCE.

a. Received diagnostic packets shall be logged using standard CDCNET logging facilities.

b. No other action shall be taken in response to a diagnostic packet. No ULP indications shall be sent, no response shall be sent to the X.25 peer, no requests shall be made to the HDLC layer, and no changes shall be made to the internal state variables for the associated virtual circuit.

3.2.3.3 Log Messages

X.25 shall provide log messages for operations and network analysis using the Log M-E interface. The standard X.25 modules currently generate several messages. The following additional messages shall be generated to support the DDN interface.

3.2.3.3.1 X.25 Packet Level Log Messages

[L O G M E S S A G E P U R P O S E

[

[This message indicates that the X.25 Packet Level received a
[diagnostic packet.

[

[A C T I O N R E Q U I R E D

[

[None.

[

[D E S C R I P T I V E M E S S A G E

[

M A S K	L O G _ M E S S A G E _ B U F F E R			
fixed text	type	value	description	
See Mask 1 below	char	1..31	X.25 Interface Name	
See Mask 2 below	int	0..255	Diagnostic code	
See Mask 3 below	int	0..3	Number of explanation bytes	
See Mask 4 below	binary octets	3 bytes	Diagnostic explanation bytes	

[mask1 - 'X.25 Diagnostic packet received on Interface Name ='
[mask2 - 'Containing Diagnostic code ='
[mask3 - 'with number of Explanation bytes ='
[mask4 - ', Explanation ='
[

```
[ T E M P L A T E  I D s
[
[         XPETEMP
[
[
[ L O G  M E S S A G E  ID  and  L O G  M E S S A G E  A T T R I B U T E S

CONST
  xpl$diag_packet_received = min_log_message_id + 1147;
[ EL
```

3.2.3.4 Intranet 3A

X.25 shall provide the interface to the ULP via the CDNA Intranet 3A module. This module shields the details of the specific network solution from the ULP. No changes are required to Intranet to support DDN X.25. However the architecture_type feature, which is not a formal part of the CDCNET release for the DDN baseline (release 1.1), is required.

3.2.3.5 Internet Protocol

X.25 shall provide a network solution interface to the Internet Protocol. This interface is provided transparently via the CDNA Intranet 3A module. No changes are required to X.25 to support the IP ULP.

3.2.3.6 HDLC

X.25 shall use the CDCNET HDLC SSR ULP interface for the link-level interface with the remote X.25 peer. No changes are required to HDLC to support DDN X.25.

3.2.4 Internal Interfaces

The following subsections present the detailed design for the modifications to CDCNET X.25. Changes to common data definitions are given in section 3.2.6. In some cases, the design for the entire module to be changed is given. New lines are marked by a plus sign (+), modified lines are marked by an asterisk (*), and lines that have simply changed indentation (for example, because an IF statement was inserted) are marked by a minus sign (-). In other cases, only a fragment of the existing module along with the identified changes, or a description of the area to be changed, will be given; in these cases, a deck/line number reference is given in brackets.

The modifications are made to three separate CDCNET modules:

1) X25 Network Solution Command Processor Module [XNMCMDP]
2) X25 Network Solution Server Module [XNMX25N]
3) X25 Packet Level Module [XPMPLC]

3.2.4.1 X25 Network Solution Command Processor Module

The X25 network solution command processor is modified to accept the new DDN precedence parameter. The following procedures are modified:

1) cmd_define_x25_net

3.2.4.1.1 cmd_define_x25_net Modifications

The following changes are made to this procedure:

a) Before the parameter loop /defxn_parm_loop/, and after parm_index is initialized, the ddn_precedence item will be initialized to 'unspecified'. [XNMCMDP.434]

b) At the end of the parameter case statement (after the parm_at case), a new case statement will be added called parm_dxp. The body of the case will set the ddn_precedence item to the value of the parameter. [XNMCMDP.488]

3.2.4.2 X25 Network Solution Server Module

The X25 network solution server is modified to handle the DDN Standard and DDN precedence facilities, and to manage CDNA-specific headers, for DDN Standard networks. The following procedures are modified:

1) Data Structures
2) send_call_request
3) x25ns_layer_management
4) x25ns_connection_management
5) x25ns_data_indication
6) process_x25ns_event
7) build_facilities (new procedure)
8) validate_facilities (new procedure)

3.2.4.2.1 Data Structure Modifications

The following changes are made to X.25 network solution data structures:

a) Add a new data structure to handle facilities processing. [XNMX25N.129]

```
CONST
  private_facility_marker = 0;
  fac_ddn_precedence_code = 08(16);
  fac_ddn_precedence_parm = 0;
  fac_ddn_standard_code = 04(16);
  fac_ddn_standard_parm = 1;
  fac_fast_select_code = 1;
```

```
    fac_fast_select_parm = 80(16);
    fac_reverse_charge_code = 1;
    fac_reverse_charge_parm = 1;

  VAR
    facilities: [STATIC] packed record
      length: 0 .. 255,
      list:   array [1 .. 32] OF packed record
        code:      0 .. 255,
        parameter: 0 .. 255,
      recend,
    recend;
```

b) Add an additional clear-request diagnostic code for facility processing. [XNMX25N.86]

```
    cr_none = 0,
+   cr_invalid_facility = 41(16);
    cr_peer_first = 90(16),
      .
      .
      .
      etc.
```

3.2.4.2.2 send_call_request Modifications

The following changes are made to this procedure:

a) A new procedure called build_facilities is called to build the facilities block, which can include the DDN Standard and Precedence facilities.

b) The CDNA call request data is not sent if the packet is destined for a foreign network (for example, DDN Standard). [XNMX25N.679]

```
Begin Procedure
    Call append to add remote dte address to packet
*   Call build_facilities to handle accept_pdn_charges and DDN
        facilities
    Call append to add the facilities block to the data packet
    Set the protocol id into the user data
    Set the system and network id into the user data
+   IF architecture_type does not include cdna THEN
+     Set user data length to one byte (for protocol id)
+   ELSE
+     Set user data length to size of user_data record,
+         to include cdna connection information
*   Call append to add the user data to the packet,
*       using user data length
        .
        . [remainder of procedure]
        .
PROCEND;
```

3.2.4.2.3 x25ns_layer_management Modifications

The following changes are made to this procedure:

a) If the user data does not match the expected cdna length, an incoming call request currently is rejected. This is changed so that the list of networks (Network Information Block chain, nib) is searched for a foreign (that is, not CDNA architecture) network that matches the calling DTE address and protocol id. If one is found, the facilities are checked for correctness and the call is accepted. [XNMX25N.926]

```
      CASE of indication type

   [  ======================
      = pl_call_indication =
   [  ======================

        Call get_first_byte to extract address length from block
   *    Call strip to get calling dte address and save locally
   +    Call strip to get called dte address
        Call get_first_byte to get facilities length
   *    Call strip to get facilities block, saving in the new global
   *        facilities record
   +    Call strip to get user data block, saving in global user
            data record

   *    IF length of user data is not equal to expected cdna size
            THEN
          Search list of nib's for network with id that matches
              user data
          IF nib NOT found THEN
            Set clear reason to cr_network_ide_not_defined
          ELSE
            Verify network is X25, reverse charges are correct, sap
                is
                correct

   +    IF call request does match cdna requirements THEN
   +        [that is, if clear_reason is not cr_none]
   +      Loop through nib chain, searching for network with the
   +          following characteristics:
   +            a. calling dte address matches nib remote dte
                       address
   +            b. user data protocol id matches nib protocol id
   +            c. nib architecture_type does not include cdna
```

```
+       End loop
+       IF network not found THEN
+         Set clear reason to cr_incorrect_data_length
+       ELSE network found
+         Call validate_facilities to evaluate inbound facilities
+         IF facilities are valid THEN
+           Set the network id in user data record to value in lib
+           Set the system id in user data record to broadcast
                address
+              since actual system id is unknown - this will
                   cause
+              all call collisions from foreign architecture
                   networks
+              to always resolve in favor of the incoming call.
+
      IF clear_reason is cr_none THEN
        .
       . (remainder of procedure)
        .
  PROCEND;
```

3.2.4.2.4 x25ns_connection_management Modifications

The following changes are made to this procedure:

a) A 3A header is created and prefixed to the data buffer before sending it to Intranet, if the network is not a cdna architecture. [XNMX25N.1050]

```
  Begin Procedure
      Set pointer to link_interface_block (lib)
      IF indication is a data indication and link is active THEN
+       IF architecture_type does not include cdna THEN
+         Set local 3A header DSAP and SSAP fields to DDN IP
```

```
+             protocol id (96)
+          Call prefix to add 3A header to buffer
          Call 3A to deliver data buffer
        ELSE
          .
          . [remainder of procedure]
          .
      PROCEND;
```

3.2.4.2.5 x25ns_data_indication Modifications

The following changes are made to this procedure:

a) The 3A header is stripped from the data buffer before sending it to the packet level module, if the network is not a cdna architecture network (that is, DDN Standard network). [XNMX25N.1174]

```
Begin procedure
    Get data buffer pointer
    IF link is not active THEN
      Discard message and issue log message
    ELSE
+     IF architecture_type does not include cdna THEN
+       Call strip to remove 3A header
      Issue packet level data request
      IF data request rejected THEN
        Issue log message
  PROCEND;
```

3.2.4.2.6 process_x25ns_event Modifications

The following changes are made to this procedure:

a) A call is made to the new build_facilities procedure for the case when a call_accept packet is sent. [XNMX25N.1297]

```
[   ========================
    = nsa_accept_call_ns_up =
[   ========================

      Set packet level request type
      Set packet level CEPID
      Clear D-bit in request
*     Call build_facilities to handle DDN facilities
      Set request block size
      Issue call_accept request
         .
         . [remainder of procedure]
         .
PROCEND;
```

3.2.4.2.7 New Procedure build_facilities

A new procedure is defined that builds facilities to be sent in a Call Request or Call Confirm packet.

```
[# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *]
[                                                                       ]
[     PROCEDURE NAME:  B U I L D _ F A C I L I T I E S                   ]
[                                                                       ]
[     PURPOSE:                                                          ]
[       Build facilities block for Call Request/Confirm packets.        ]
[                                                                       ]
[     DESCRIPTION:                                                      ]
[       The facilities block for a packet is generated for Call         ]
[       Request and Call Confirm packets, based upon network            ]
[       configuration information.                                      ]
[                                                                       ]
[     INPUTS:                                                           ]
[       lib_ptr          ^x25ns_lib_type        P1: lib pointer         ]
[       action           range_of_x25ns_actions P2: action to take      ]
[                                                                       ]
[     PROCEDURES REFERENCED                                             ]
[                                                                       ]
[     OUTPUTS:                                                          ]
[       facilities                        GL: global facilities block   ]
[                                                                       ]
[* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *]
```

```
  PROCEDURE build_facilities ( [
        lib_ptr: ^x25ns_lib_type; [
        action: range_of_x25ns_actions);
```

```
[$

    VAR
      count: 0 .. 255,
      private_marker: 0 .. 255;

   [
[ * * *  BUILD STANDARD X25 FACILITIES.
[

    Set local facility count to zero
    IF action is nsa_accept_call_ns_up THEN
      IF accept_pdn_charges THEN
        Set facilities.list[count].code to fac_reverse_charge_code
        Set facilities.list[count].parm to fac_reverse_charge_parm
        Increment count

[
[ * * *  BUILD PRIVATE X25 FACILITIES, IF ANY, AFTER FACILITIES MARKER.
[

    Set facilities.list[count].code to private_facility_marker
    Set facilities.list[count].parm to private_facility_marker
    Set private_marker to count
    IF pdn name is "DDN_STANDARD" AND protocol id is ddn_standard THEN
      Set facilities.list[count+1].code to fac_ddn_standard_code
      Set facilities.list[count+1].parm to fac_ddn_standard_parm
      Increment count
    IF ddn precedence is NOT unspecified THEN
      Set facilities.list[count+1].code to fac_ddn_precedence_code
      Set facilities.list[count+1].parm to fac_ddn_precedence_parm +
          value of configured ddn precedence
      Increment count
```

```
        IF count is not equal to private_marker THEN
            [that is, private facilities exist]
          Increment count
        Set facilities.length to count * 2 + 1

      PROCEND build_facilities;
```

3.2.4.2.8 New Procedure validate_facilities

A new procedure is defined that validates facilities received in a Call Request packet.

```
[# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *]
[                                                                     ]
[    PROCEDURE NAME:  V A L I D A T E _ F A C I L I T I E S          ]
[                                                                     ]
[    PURPOSE:                                                         ]
[      Validate facilities received in a Call Request packet.         ]
[                                                                     ]
[    DESCRIPTION:                                                     ]
[      The facilities block for a packet received in a Call           ]
[      Request is compared to configured information.  A clear        ]
[      reason code is returned if any facilities conflict             ]
[      or are unrecognized.                                           ]
[                                                                     ]
[    INPUTS:                                                          ]
[      lib_ptr          ^x25ns_lib_type P1: lib pointer              ]
[      facilities                       GL: global facilities block ]
[                                                                     ]
[    PROCEDURES REFERENCED                                            ]
[                                                                     ]
[    OUTPUTS:                                                         ]
[      clear_reason    0 .. 255       P2: clear reason code          ]
[                                                                     ]
[* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *]
```

```
  PROCEDURE build_facilities ( [
        lib_ptr: ^x25ns_lib_type; [
    VAR clear_reason: 0 .. 255);

[$

    VAR
      count: 0 .. 255,
      private_facilities: boolean;

    Set clear_reason to cr_none
    Set private_facilities to FALSE
    Loop through facilities block, using count as the index from
        0 to (facilities.length-1)/2 DO

[
[ * * *  VALIDATE PRIVATE FACILITIES.
[

      IF private_facilities THEN
        CASE of facilities.list[count].code

[       ==========================
        = fac_ddn_standard_code =
[       ==========================

          IF pdn name is NOT "DDN_STANDARD" OR
               protocol id is NOT ddn standard OR
               architecture type does not include dod OR
               architecture type includes cdna THEN
            Set clear_reason to cr_invalid_facility

[       ============================
        = fac_ddn_precedence_code =
[       ============================
```

```
            IF configured precedence is NOT unspecified AND
                 configured precedence is NOT equal to facility value THEN
              Set clear_reason to cr_invalid_facility

          ELSE
            Set clear_reason to cr_invalid_facility

          CASEND;

[
[ * * *  VALIDATE STANDARD FACILITIES.
[

        ELSE
          CASE of facilities.list[count].code

[         ============================
          = private_facility_marker =
[         ============================

            Set private_facilities to TRUE

[         ================================================
          = fac_reverse_charge_code, fac_fast_select_code =
[         ================================================

            ; [ These facilities are handled by X25 packet level

          ELSE
            Set clear_reason to cr_invalid_facility

          CASEND;

    PROCEND validate_facilities;
```

3.2.4.3 X25 Packet Level Module

The X25 packet level module is modified to log diagnostic packets. The following procedures are modified:

1) data structures
2) log_pl_error
3) ssr_data_ind_proc

3.2.4.3.1 Data Structure Modifications

The following changes are made to X.25 packet level data structures:

a) Add log message to list of message constants. [XPMPLC.1197]

```
        xpe_circuit_statistics = 6,
+       xpe_diag_packet_received = 7,
*       xpe_last = xpe_diag_packet_received;
```

b) Add new log message case to log_record_type. [XPMPLC.1221]

```
+       = xpe_diag_packet_received =
+         diag_code: 0 .. 255,
+         expl_size: 0 .. 3,
+         explanation: 0 .. 0ffffff,
        casend,
      recend;
```

c) Add new log message preset to pl_template_id. [XMPMPLC.1233>AC1D758.109]

```
*           [xpe$circuit_statistics, xpl$circuit_statistics,
               FALSE, TRUE], [
+           [xpe$diag_packet_received, xpl$diag_packet_received,
               TRUE, TRUE]];
```

3.2.4.3.2 log_pl_error Modifications

The following changes are made to this procedure:

a) A new case item is added at the end of the case statement for the new log message. [XPMPLC.4863]

```
+[      ============================
+       = xpe_diag_packet_received =
+[      ============================
+
+         Call gen_data_field to put diagnostic code into log
             buffer
+         Call gen_data_field to put explanation length into log
             buffer
+         Call gen_data_field to put explanation bytes into log
             buffer
+
        ELSE
          RETURN
```

3.2.4.3.3 ssr_data_ind_proc Modifications

The following changes are made to this procedure:

a) The case item for diagnostic packets is extended to call log_pl_error to log the diagnostic packet information.

```
[    ==================
     = diagnostic_pkt =
[    ==================
```

```
+       Call strip to get diagnostic code from packet, and put
            into log record.
+       Compute explanation size from message size (msg_size)
            minus fixed
+           header size (4 octets), and put into log record.
+       Call strip to get explanation bytes from packet, using
            explanation
+           size, and put into log record.
+       Call log_pl_error to log diagnostic packet.
        Call release_message to discard packet
        RETURN
```

3.2.5 Errors and Error Recovery

Standard CDCNET X.25 HDLC, Packet Level, and Network Solution level error recovery shall be employed. Support of diagnostic packets shall be as specified in 3.2.3.2.5.

3.2.6 Data Structures

The following subsections define the changes required to common CDCNET data structures to support DDN Standard X.25. The data structures to be changed, and the deck they reside in, are:

1) X.25 DTE Control Block (CMDLIBX)
2) PDT for Define_X25_Net (XNDDFXN)

3.2.6.1 X.25 DTE Control Table (CMDLIBX.31)

One item will be added to the X.25 DTE Control Table. The item will contain the DDN Precedence, and will be set by the command processor using the value specified in the DEFXN command; a special value is defined to recognize when the parameter is not specified. The new definitions are shown below.

```
TYPE
  x25_dte_table_type = record
      .
      .  (existing definitions)
      .
    ddn_precedence: (null, low, medium, high, unspecified), [ddn
       precedence
  recend;
```

3.2.6.2 PDT for Define_X25_Net (XNDDFXN.3)

A new parameter description will be added to the end of the parameter descriptor table (PDT) for the DEFXN command. The new description is shown below.

```
[ pdt cmd_defxn_pdt (
      .
      . (existing definitions)
      .
[   ddn_x25_precedence,dxp: integer 0..3 = $optional
[   )
```

The PDT must be re-run through the PDT pre-processor to re-generate the CYBIL definitions in the remainder of XNDDFXN; the new definitions must replace the existing ones.

3.2.7 Equipment Configuration

The X25 module, and the above specified modifications, are designed to run in a CDCNET DI that is configured with at least one CIM and one LIM which provide the physical X.25 connection. Although the X25 module and modifications are not designed for a specific physical interface, the interface specified in section 3.1 at a minimum shall be supported.

3.2.8 Installation

The following subsections define the installation procedures and parameters for the X.25 modifcations CPCI.

3.2.8.1 X.25 Installation Procedure

The X.25 modifications are placed into the CDCNET system using the standard DDN/CDCNET update procedures defined in section 3.10.8. The PL used as input to the procedures is X25nnnn, where nnnn is the CDCNET level number to which the transmittal applies. The installation procedure must be run after any other CDCNET modifications installation procedures, as specified in section 3.10.

3.2.8.2 X.25 Installation Parameters

No installation parameters are defined for the X.25 modifications.

3.2.8.2 X.25 Network Configuration

The following subsections define the DDN-specific requirements for the configuration of the X25 interface.

3.2.8.1.2 Link Level Operational Parameters (XHIF 2.3)

The Define_X25_Trunk command shall be used to specify the X25 link level operational parameters (refer to ARH6785, Network Definition, sec 9.5.5.2).

a. The mode parameter shall be set to DTE.

b. The max_unack_frames (K) parameter shall be set to 7.

c. The pf_recovery_timer (T1) parameter shall be set to 3000 ms.

d. The retransmission_limit (N2) parameter shall be set to 20.

e. The trunk_speed parameter shall be set to 9600 or 56000, depending upon the link speed.

f. The clocking parameter shall be set to EXTERNAL. The LIM cannot provide clocking for a 56Kbps link.

3.2.8.2.2 Packet Level Operational Parameters (XHIF 2.1.1, 2.2)

The Define_X25_Interface and Define_X25_Network commands shall be used to specify the X25 packet level operational parameters (refer to ARH6785, Network Definition, secs 9.5.4, 9.5.5.2).

a. The remote_dte_address, local_dte_address, pvc_range, inonly_range, twoway_range, outonly_range, and default_window_size parameters shall be set to the values assigned by DCA.

b. The packet_sequence_numbering parameter shall be set to NORMAL.

c. The default_packet_size parameter shall be set to 1024 octets to efficiently accomodate the IP maximum packet size.

3.2.8.2.3 CDNA Operational Parameters

The Define_X25_Interface and Define_X25_Network commands shall be used to specify the X25 CDNA operational parameters (refer to ARH6785, Network Definition, secs 9.5.4, 9.5.5.2).

a. The cost parameter shall be set to 06FA(16) for a 56Kbps link, or to 28B1(16) for a 9.6Kbps link.

b. The relay_allowed parameter shall be set to NO.

c. The routing_info_network parameter shall be set to NO.

d. The ddn_x25_precedence, protocol_id, architecture_type, and pdn_name parameters shall be as defined in section 3.2.3.1.

3.2.8.2.4 Configuration Example

Following is a sample configuration excerpt for an X.25 connection to a PSN using DDN Standard X.25.

```
Define_X25_Trunk                    " Trunk for DDN Standard X.25 link "..
  lim = 7,..
  port = 0,..
  trunk_name = DDN_Interface,..
  mode = DTE,..
  max_unack_frames = 7,             " X.25 parameter K  "..
  pf_recovery_timer = 3000,         " X.25 parameter T1 "..
  retransmission_limit = 20,        " X.25 parameter N2 "..
  trunk_speed = 56000,..
  clocking = EXTERNAL

Define_X25_Interface                " Packet Level for DDN X.25 link "..
  trunk_name = DDN_Interface,..
  public_data_network = DDN_STANDARD,..
  interface_name = DDN_Interface,..
  local_dte_address = 100436(16),   " DCA assigned      "..
  packet_sequence_numbering = NORMAL,..
  pvc_range = 1..5,                 " DCA assigned      "..
  outonly_range = 4090..4095,       " DCA assigned      "..
  default_window_size = 7,          " DCA assigned      "..
  default_packet_size = 1024,..
  start = YES
```

```
Define_X25_Net                      " Network Solution for DDN X.25 "..
  trunk_name = DDN_Interface,..
  remote_dte_address = 52A8C0(16),  " DCA assigned       "..
  network_id = 150,                 " CDCNET network id  "..
  network_name = DDN_Interface,..
  cost = 06FA(16),                  " assumes 56Kbps     "..
  relay_allowed = NO,               " not a CDCNET net   "..
  routing_info_network = NO,        " not a CDCNET net   "..
  network_protocol_id = CC(16),     " DDN Standard pid   "..
  accept_pdn_charges = NO,..
  architecture_type = dod,          " not a CDCNET net   "..
  ddn_x25_precedence = 0,..
  start = YES
```

3.2.8.2 X.25 Modifications Transmittal

3.2.8.2.1 Transmittal Files

The following files are provided in the transmittal:

a. X25 Modifications Program Library (X25nnnn). This is an SES MODIFY formatted program library containing the X25 modifications. The PL uses a slash-prefix (SLASHPL) to maintain modsets to be applied against the CDCNET program library.

3.2.8.2.2 PL Structure

The X25 modifications PL follows the PL conventions given in section 3.10.8. It contains the following decks in the indicated order:

INSTALL	Deck to contain special installation procedure, empty for X25
EDIT	Deck containing EDIT directives required for X25 installation
-MODS-	Empty deck marks beginning of modsets
DDNX25	Deck containing front matter for DDN X25 feature modset

HISTORY	Deck containing modifications to CDCNET HISTORY deck
.....	One deck for each CDCNET PL deck that is modified. The deck is named the same as the deck to be modified, and contains all modset lines for that deck. The decks appear in alphabetical order.
ENDX25	Deck containing end matter for X25 modset
-ENDM-	Empty deck marks end of modsets
-DECKS-	Empty deck marks beginning of new decks, none supplied for X25
-ENDD-	Empty deck marks end of new decks

3.2.8.2.3 Dependencies

No other DDN CPCI transmittals may depend upon the X25 transmittal.

The X25 transmittal depends upon the CDCNET modifications transmittal (if any).

3.3 Internet Protocol (IP)

3.3.1 IP Abstract

The IP protocol is designed to transmit and receive packets of information across the DDN and networks connected to the DDN. To communicate across multiple networks, IP supports a global addressing system and accommodates differences in maximum packet sizes allowed by networks. IP has no mechanisms to promote data reliability, flow control, or sequencing. It provides only the basic functions necessary to deliver a block of data.

3.3.2 IP Description

The IP CPCI is specified in a separate document entitled DoD Internet Protocol ERS. That document defines an IP design and implementation that is based upon the Berkeley 4.2 UNIX IP.

3.3.3 External Interfaces

Refer to the DoD Internet Protocol ERS, Section 3.0.

3.3.4 Internal Interfaces

Refer to the DoD Internet Protocol IDS, Section TBS.

3.3.5 Errors and Error Recovery

Refer to the DoD Internet Protocol ERS, subsection 3.5

3.3.6 Data Structures

Refer to the DoD Internet Protocol ERS, Section 8.0

3.3.7 Equipment Configuration

DDN IP requires no special equipment configuration.

3.3.8 Installation

Refer to the DoD Internet Protocol ERS, Section 7.0

3.4 Transmission Control Protocol (TCP)

3.4.1 TCP Abstract

The TCP protocol is designed to provide reliable communication between pairs of processes in logically distinct hosts on networks and sets of interconnected networks. It provides connection-oriented data transfer that is reliable, ordered, full duplex, and flow controlled in an environment where loss, damage, duplication, or misordered data, and network congestion can occur.

3.4.2 TCP Description

The TCP CPCI is specified in a separate document entitled DoD Transmission Control Protocol ERS. That document defines a TCP design and implementation that is based upon the Berkeley 4.2 UNIX TCP.

3.4.3 External Interfaces

Refer to the DoD Transmission Control Protocol ERS, Section 3.0.

3.4.4 Internal Interfaces

Refer to the DoD Transmission Control Protocol IDS, Section TBS.

3.4.5 Errors and Error Recovery

Refer to the DoD Transmission Control Protocol ERS, subsection 3.6.

3.4.6 Data Structures

Refer to the DoD Transmission Control Protocol ERS, Section 8.0.

3.4.7 Equipment Configuration

DDN TCP requires no special equipment configuration.

3.4.8 Installation

Refer to the DoD Transmission Control Protocol ERS, Section 7.0.

3.5 TELNET

3.5.1 TELNET Abstract

The TELNET protocol is designed to provide communication between hosts and terminals of different vendors. It is based upon the concept of a Network Virtual Terminal, the principle of negotiated options, and a symetric view of terminals and processes. During a session between a terminal and a host, TELNET resolves differences between the actual characteristics of the terminal and those of NVT.

3.5.2 TELNET Description

The TELNET CPCI is specified in a separate document entitled DoD TELNET/BSD Gateway ERS. That document defines a TELNET and CDNA/TELNET Gateway design and implementation that is based upon the Berkeley 4.2 UNIX TELNET.

3.5.3 External Interfaces

Refer to the DoD BSD Gateway/TELNET ERS.

3.5.4 Internal Interfaces

Refer to the DoD BSD Gateway/TELNET IDS.

3.5.5 Errors and Error Recovery

Refer to the DoD BSD Gateway/TELNET ERS.

3.5.6 Data Structures

Refer to the DoD BSD Gateway/TELNET ERS.

3.5.7 Equipment Configuration

DDN TELNET requires no special equipment configuration.

3.5.8 Installation

Refer to the DoD BSD Gateway/TELNET ERS.

3.6 File Transfer Protocol (FTP) (RFC-765)

3.6.1 FTP Abstract (RFC-765)

The purpose of FTP is to provide users of possibly dissimilar computers the capability to exchange files of computer programs and data without requiring knowledge of each computer's file storage system.

The FTP will provide server and user functions for DDN users. The server function will provide remote DDN users and applications with access to the CYBER host file system. The user function will provide CYBER host terminal users and DDN TELNET users with access to file systems on remote hosts.

3.6.2 FTP Description (RFC-765, pg 248)

Figure 3.6-1 illustrates a model of the FTP service in which the user initiates a TELNET connection to the server. FTP commands and responses are sent and received via the TELNET connection while file transfers are accomplished via the data connection.

At the initiation of the user, standard FTP commands are generated by the User-Protocol Interpreter (PI) and transmitted to the server process via the TELNET connection. The user may also establish a direct TELNET connection to the Server-FTP, from a TAC terminal, for example, and generate FTP commands himself, bypassing the User-FTP process. Replies are sent from the Server-Protocol Interpreter (PI) over the TELNET connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The User-Data Transfer Process (DTP) will "listen" on the specified data port, and the Server-Data Transfer Process (DTP) will initiate the data connection and data transfer in accordance with the specified parameters.

It should be noted that the data port need not be in the same host that initiates the FTP commands via the TELNET connection, but the user or his User-FTP process must ensure a "listen" on the specified data port. It should also be noted that the data connection may be used for both sending and receiving and that FTP must support transfer of files between two hosts, neither of which is the user's host. The latter case is illustrated in Figure 3.6-2 where TELNET connections are established between two server FTPs. In this figure, control information is passed to the User-PI but data is transferred between the two Server-DTPs.

The CYBER FTP implementation consists of four components: the FTP Control Statement, the FTP Application Interface routines, the FTP Protocol Interpreter (PI), and the FTP File Server. Figure 3.6-3 depicts the FTP components and their interfaces. The FTP protocol interpreter implements both User and Server PI functions. A single FTP PI exists in a CYBER, so many user and server connections must be multiplexed. The command-response interface to FTP PI can be from TCP connections for access by remote FTP processes or from NAM A-A connections for access by TELNET users and local application programs. The FTP Application Interface (AI) routines provide the NAM A-A interface for application programs. Actual data transfer and file handling is performed by FTP File Servers (FS).

The FTP Control Statement (CS) component uses the FTP AI routines to provide a job-level (batch or interactive) interface to FTP.

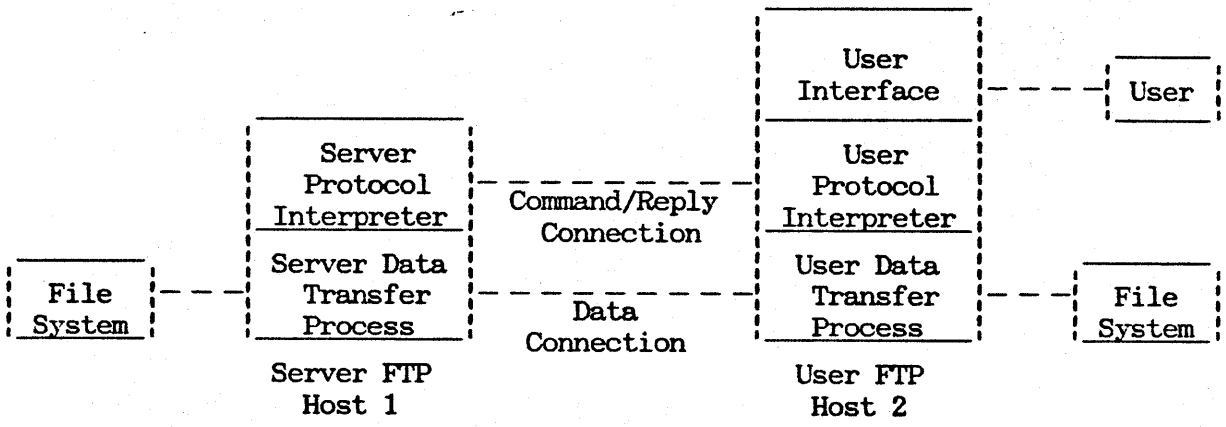


Figure 3.6-1. Model for FTP Use (Two Hosts)

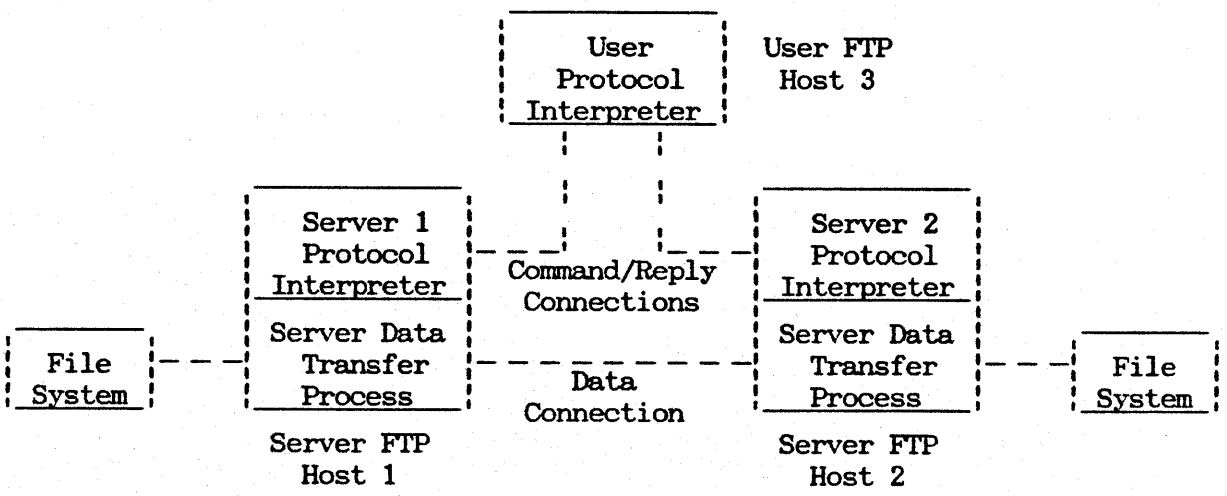


Figure 3.6-2. Model for FTP Use (Three Hosts)

The various FTP components communicate via NAM connections that are depicted in Figure 3.6-3 and summarized below:

1) FTP Application I/F to Protocol Interpreter. This connection is initiated by the AI and terminated by the PI. One connection exists for each instance of the Application I/F, which can be resident either with an FTP Control Statement or with a user application. Many instances of both the control statement and user applications are possible.

2) FTP File Server to Protocol Interpreter. This connection is initiated and terminated by the PI. One connection exists for each instance of the File Server.

3) FTP TCP Data Connection to remote FTP. This connection can be initiated and terminated by either the local FTP File Server or by a remote FTP, at the request of the PI. The NAM connection is created indirectly via the TCP/IP C170 DoD Interface routines and is between the File Server and the NP DoD Gateway residing in the MDI. One connection exists for each FTP file server that has a TCP data connection with a remote FTP.

4) FTP TELNET Command Connection to remote FTP. This connection can be initiated and terminated by either the local FTP Protocol Interpreter or by a remote FTP, at the request of the PI. The NAM connection is created indirectly via the TCP/IP C170 DoD Interface routines and is between the Protocol Interpreter and the NP DoD Gateway residing in the MDI. One connection exists for each TELNET connection between FTP PI and a remote FTP.

5) FTP to DDN Supervisor. This connection is initiated and terminated by FTP PI and FTP File Server. The NAM connection is created indirectly via the Supervisor C170 DoD Interface routines for the name server and error logging services. One connection exists for the PI and for each instance of a File Server.

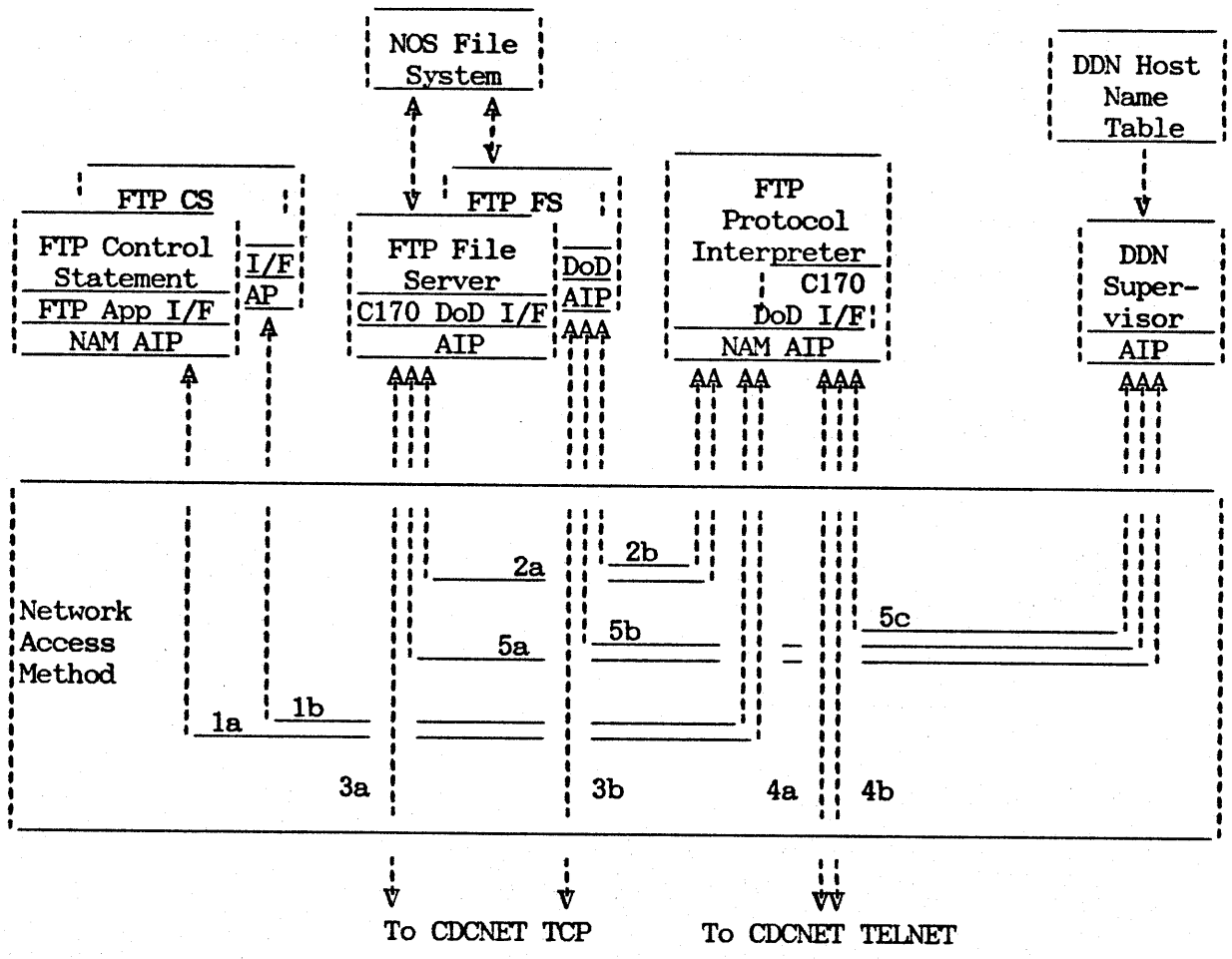


Figure 3.6-3. FTP Functional Relationships

3.6.2.1 File Transfer Functions (RFC-765, pg 263)

File transfer functions shall be initiated by the user sending FTP commands to the server and interpreting the returned replies, and the server shall be responsible for interpreting and processing commands and sending replies and data to the user.

3.6.2.1.1 FTP Command Syntax (RFC-765, pg 284)

The FTP commands are TELNET ASCII character strings terminated by the TELNET end-of-line code and are partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. The syntax of all FTP commands shall be as specified in a. through f. below.

a. FTP commands shall begin with a command code of from one to four alphabetic ASCII characters. (RFC-765, pg 284)

b. Upper and lower case alphabetic characters shall be recognized equally in the command code. (RFC-765, pg 284)

c. An argument field, if present, shall follow the command code and shall be separated from the command code by one or more ASCII spaces. (RFC-765, pg 284)

d. The argument field shall consist of a variable length ASCII character string. (RFC-765, pg 284)

e. Upper and lower case alphabetic characters shall be recognized equally in the argument field. (RFC-765, pg 284)

f. The FTP command shall be terminated with the TELNET end-of-line code consisting of the ASCII character sequence of Carriage Return and Line Feed. (RFC-765, pg 284)

g. No action shall be taken by the FTP server until the TELNET end-of-line code is received. (RFC-765, pg 284)

NOTE: The actual command codes, argument fields, and syntax of all FTP commands to be implemented are stated in subsections 3.6.2.1.1.1 through 3.6.2.1.1.3 below.

3.6.2.1.1.1 Access Control Commands (RFC-765, pg 263)

Access Control Commands define users' access privileges to the use of a system and to the files in that system. They are used to prevent unauthorized and accidental use of a system and files and are enforced by the server FTP. These commands are described below.

NOTE: In the COMMAND FORMAT subsections a.1 through e.1, items enclosed in brackets [] have the following meaning:

[SP] - represents one ASCII space.

[...] - represents a command argument.

[CRLF] - represents the TELNET end-of-line character string of Carriage Return Line Feed.

a. USER NAME COMMAND (RFC-765, pg 264)

The user name command shall be a TELNET string that identifies the user to the server FTP for access to the NOS file system.

1. COMMAND FORMAT - USER[SP][FAMILY][,USERNAME][,PASSWORD] (RFC-765, pg 285)

USER - The command code shall be the character string "USER" upper and/or lower case ASCII characters.

FAMILY - The FAMILY argument is optional. If present, it shall consist of one to seven alphanumeric characters. If omitted, the default family is used.

USERNAME - The USERNAME argument shall be a NOS user number that shall consist of one to seven alphanumeric characters. Note that whether or not the FAMILY argument is present, USERNAME shall be preceeded by a comma(,).

PASSWORD - The PASSWORD argument is optional. If present, it shall be a NOS password that shall consist of one to seven alphanumeric characters and shall be preceeded by a comma(,).

If password is omitted, a 331 response is issued with the text 'need password for login' and the user must enter the PASS command. If password is supplied, a 230 or 332 response is issued. Use of the PASS command instead of direct entry on the USER command may provide more secure entry, since the user FTP may supply overprint, no-echo, or other protection. CYBER user interactive FTP will provide overstrike characters, similar to the NOS password command, but will not attempt to change echo (since most CYBER systems use local terminal echo). Thus, the protection offered by a CYBER is only useful for hard-copy terminals; it is suggested that a CRT user enter the terminal's screen-clear character following the password if protection is needed.

The USER command can usually be entered at any time, but doing so will require re-entry of the PASS and ACCT commands. However, if the site has disabled secondary user commands, the USER command can only be entered once.

If the family/username/password combination is incorrect, a 530 response is returned. Entry of any command other than USER/PASS/ACCT/HELP/REIN/QUIT/ ABOR/ NOOP will result in a 532 response. If a correct combination is entered but the security count is exhausted, a 431 multi-line response is sent with a 'security count exhausted' message and FTP closes the TELNET

connection; if the command was sent from a local user, the NAM connection with that user is also terminated.

b. PASSWORD COMMAND (RFC-765, pg 264)

The Password command shall be a TELNET string identifying the user's password to the server FTP for access to the NOS file system.

1. COMMAND FORMAT - PASS[SP][PASSWORD][CRLF]

PASS - The command code shall be the character string "PASS" in upper and/or lower case ASCII characters.

PASSWORD - The PASSWORD argument shall be a NOS password that shall consist of one to seven alphanumeric characters.

The password is always required on a NOS system. If not supplied with a USER command, then it must be supplied with the PASS command. If the password was supplied on the USER command, a 202 response with the text 'command superfluous, password already given' is issued. The PASS command can be entered only immediately after a USER command; if it is entered at any other time, a 503 response is returned.

A user or peer FTP has only three (installation changeable) chances to enter a correct family/user/password combination when first logging into FTP. After the third failure, a 431 multi-line response is sent with a 'user retry limit exceeded' message and FTP closes the TELNET connection; if the command was sent from a local user, the NAM connection with that user is also terminated. The retry limit does not apply to subsequent (that is, secondary) USER commands issued in the same FTP session; these commands are subject only to the user's security count as described above. The site can disable the retry limit (that is, define an unlimited retry) by setting the installation parameter to zero.

Note that if the user has different batch and interactive passwords, the BATCH password must be entered.

c. ACCOUNT COMMAND (RFC-765, pg 264)

This command shall be used to identify the user's account and to record on the NOS account dayfile information regarding resources used.

1. COMMAND FORMAT - ACCT[SP][CHARGENUM,PROJECTNUM][CRLF]

-or-

ACCT[SP][*]

(RFC-765, pg 285)

ACCT - The command code shall be the character string "ACCT" in upper and/or lower case ASCII characters.

CHARGENUM - The CHARGENUM argument shall be a NOS charge number assigned by the sit to the user and shall consist of 1 to 10 alphanumeric characters in upper and/or lower case. The comma following CHARGENUM shall be mandatory.

PROJECTNUM - The PROJECTNUM argument shall be a NOS project number assigned by the site to the user and shall consist of 1 to 20 alphanumeric characters in upper and/or lower case.

* - If the ACCT command is entered with the * parameter, the default charge is used (if one is defined and it is valid). Note that the ACCT * command is redundant since the default charge is always assumed if one exists; it is provided for consistency with existing NOS command format.

A typical NOS site will require an ACCT command. An ACCT command will not be required in the following instances:

1) The user has the charge-not-required (CCNR) bit set in his validation file entry (AACW).

2) The user has a default charge/project defined in his validation file entry. In this case, the default charge/project (if valid) are automatically used unless an ACCT command is given.

If a charge is not required, a 230 response is returned from the USER command. If a charge is not required and an erroneous ACCT command is issued (that is, invalid charge) then a 202 response is issued with the text 'command superflous, charge rejected' followed by an extended error message containing the NOS CHARGE processor diagnostic (see diagnostics, NOS v.3).

If the charge-not-restricted-to-default (CNRD) bit is set in the users validation file entry (AACW) then any charge for which the user is validated can be entered. Otherwise, only the default charge is permitted.

Although a charge can be entered at any time on a NOS system, the FTP error/response system only permits entry of an ACCT command after a USER/PASS command sequence. Entry of ACCT any time except after a USER/PASS sequence will result in a 503 response.

Charged Resource Usage. Only resources used by the file server are charged to the specified account. This means that only actual file transfer resource usage is charged; resources used by the protocol interpreter to process commands, give help, etc, are charged to an overhead account determined by the site. When the user terminates the FTP connection, all file server resources used while idle (which are small) are also charged to an overhead account. Note that any resources used by control statement processing (FTP control statement) are charged to the account under which the user is currently running; this account is changed only by the NOS USER and CHARGE statements, and is not affected by the FTP USER/ACCT commands.

After a successful user command has been entered, the permanent file restrictions associated with that account apply to the FTP file server. For example, if the user is not validated to create direct access files (CLPF in AACW validation entry) then FTP will not be able to do so either and an error will result if the user requests FTP to create a direct access file. The validations covered by this include CLPF, CSPF, CSRP, CSAF, CLTD, CS, DS, FC, FS, MS, RP, SAL, CFPX, CLFL, CWLF. The validation mnemonics are defined in the NOS Administration Handbook.

Extended USER/ACCT Responses. Several error responses are possible from an ACCT command, depending on the reason for charge failure (see diagnostics, NOS v.3). If the charge fails for reasons other than syntax errors, a multi-line 530 response will be issued with the text 'Not logged in, account rejected' followed by a line containing the specific diagnostic from the NOS charge processor (for example, 'invalid charge', 'charge expired'). A multi-line 431 response can be issued for certain user command failures also with the subsequent line consisting of the specific NOS user processor diagnostic.

Dayfile and Account Messages. Whenever file server changes user or charge numbers, the appropriate messages are issued to the dayfile and account file, in accordance with NOS accounting conventions. When the first successful user command is entered for a user, an ABUN is issued to the account dayfile; if a default charge is associated with the new user name, an ABIC message is issued to the account dayfile. An ACUN message is issued for subsequent successful user commands from the same user. Several account messages are also issued by modules called from FTP (for example, CPM and COMCCHG) and are not documented here.

Remote vs. Local Users. A remote user of a CYBER FTP system must always send a USER/PASS/ACCT sequence to FTP. A local user (that is, a user logged into the NOS system where the FTP is executing) need not enter a USER/PASS/ACCT sequence to the local FTP. If no sequence is entered, the user and charge account under which the user is currently operating are assumed; this

user/charge is specified using NOS USER and CHARGE control statements. If a local user does enter FTP USER/ACCT commands, only the file server operation is affected; the user's job (and FTP control statement execution) continues under the account specified in the NOS USER/CHARGE control statements. A USER/PASS sequence entered by a local user is treated as a secondary user command: secondary user commands must be enabled, the user retry limit does not apply, and the primary user security count is decremented for each invalid secondary username/password. Note that a local CYBER user must still supply USER/PASS/ACCT commands to the remote server FTP (or both servers, if third-party transfer is used).

d. REINITIALIZE COMMAND (RFC-765, pg 265)

This command shall cause the server FTP to complete any file transfer in progress, flush all I/O and account information, and terminate the effects of the last USER command.

1. COMMAND FORMAT - REIN[CRLF] (RFC-765, pg 285)

   REIN - The command code shall be the character string "REIN" in upper and/or lower case ASCII characters.

   COMMAND EXAMPLE - REIN

2. The server FTP shall reset defaults and leave the TELNET connection open after terminating the user. (RFC-765, pg 265)

3. The server shall then require a USER command before additional FTP functions may be performed. (RFC-765, pg 265)

e. LOGOUT COMMAND (RFC-765, pg 265)

This command shall cause FTP to complete any file transfer in progress, terminate the user, and close the TELNET connection.

1. COMMAND FORMAT - QUIT[CRLF] (RFC-765, pg 285)

QUIT - The command code shall be the character string "QUIT" in upper and/or lower case ASCII characters.

COMMAND EXAMPLE - QUIT

3.6.2.1.1.2 Transfer Parameter Commands (RFC-765, pg 265)

The transfer parameter commands shall be used to change default parameter values from the last value specified by a transfer parameter command or, if none specified, from the standard default values stated below. There shall be no order dependency associated with transfer parameter commands except that they must all precede the FTP service commands.

NOTE: In the COMMAND FORMAT subsections a.1 through e.1, items enclosed in brackets [] have the following meaning:

[SP] - represents one ASCII space.

[...] - represents a command argument.

[CRLF] - represents the TELNET end-of-line character string of Carriage Return Line Feed.

a. DATA PORT COMMAND (RFC-765, pg 265)

The data port command shall be used to specify the data port for use in the data connection.

1. COMMAND FORMAT - PORT[SP][HOST-PORT][CRLF] (RFC-765, pg 285)

   PORT - The command code shall be the character string "PORT" in upper and/or lower case ASCII characters.

   HOST-PORT - The HOST-PORT argument shall consist of a 32-bit internet host address and a 16-bit TCT port address divided into 8-bit fields separated by commas. The value of each 8-bit field shall be a decimal number in the range of 0 to 255 represented as an ASCII numeric character. (RFC-765, pg 266,285)

   COMMAND EXAMPLE - PORT h1,h2,h3,h4,p1,p2 where h1 is the high order 8 bits of the internet host address and p1 is the high order bits of the TCT port address.

2. A FTP user shall listen on a port specified in a PORT COMMAND. (RFC-765, pg 282)

3. A FTP server shall initiate the data connection from his default data port using the port specified in a PORT COMMAND. (RFC-765, pg 282)

4. The user FTP default data port shall be port U. (RFC-765, pg 257)

5. The server FTP default data port shall be port L-1. (RFC-765, pg 257)

b. PASSIVE COMMAND (RFC-765, pg 266)

The passive command shall cause the server FTP to listen on a data port (not the default data port) and wait for a connection rather than initiate a connection upon receipt of a transfer service command.

1. COMMAND FORMAT - PASV[CRLF] (RFC-765, pg 285)

   PASV - The command code shall be the character string "PASV" in upper and/or lower case ASCII characters.

   COMMAND EXAMPLE - PASV

2. The server FTP shall provide a reply code of 227 with an internet host address and a TCP port address in the format shown in subsection 3.6.2.1.1.2.a1 above for the PORT command. (RFC-765, pg 266)

c. REPRESENTATION TYPE COMMAND (RFC-765, pg 266)

1. COMMAND FORMAT - TYPE[SP][TYPE CODE 1][CRLF]
   - TYPE[SP][TYPE CODE 2][CRLF] (RFC-765, pg 285)
   - TYPE[SP][TYPE CODE 3][CRLF]

   TYPE - The command code shall be the character string "TYPE" in upper and/or lower case ASCII characters.

2. TYPE CODE 1 - The TYPE CODE 1 argument shall specify the data type for the file transfer as ASCII with a second argument that specifies that the file contains no vertical format information, the file contains TELNET vertical format control characters, or the file contains FORTRAN vertical format control characters. The TYPE CODE 1 argument shall be recognized in upper and/or lower case as shown below. (RFC-765, pg 266)

   A[SP][N or T or C where

3. A - This is the ASCII character "A" that specifies ASCII file transfer. (RFC-765, pg 266)

4. N - This is the ASCII character "N" that specifies that the file contains no vertical format control information. This is the default. (RFC-765, pg 266)

5. T - This is the ASCII character "T" that specifies that contains TELNET vertical format control characters. (RFC-765, pg 266)

6. C - This is the ASCII CHARACTER "C" that specifies that the file contains FORTRAN vertical format control characters. (RFC-765, pg 266)

   COMMAND EXAMPLE - TYPE A T ASCII with TELNET format control
                   - TYPE A C ASCII with FORTRAN format control
                   - TYPE A N ASCII with no format control

7. TYPE CODE 2 - The TYPE CODE 2 argument shall specify the data type for transfer as Image with data sent as contiguous bits. The TYPE CODE 2 argument shall be recognized in upper and/or lower case in the format shown below. (RFC-765, pg 253, 266)

   I

   COMMAND EXAMPLE - TYPE I

8. TYPE CODE 3 - The TYPE CODE 3 argument shall specify the local byte size option. This type code is intended to facilitate file transfers between two Cyber hosts. Reference section 3.6.2.5.7 for details of this option. The TYPE CODE 3 argument shall be recognized in upper and/or lower case in the format shown below. (RFC-765, pg 253, 266)

   L[SP]60

   COMMAND EXAMPLE - TYPE L 60

d. FILE STRUCTURE COMMAND (RFC-765, pg 267)

The file structure command shall specify the file structure for file transfer.

1. COMMAND FORMAT - STRU[SP]S-CODE[CRLF] (RFC-765, pg 285)

STRU - The command code shall be the character string "STRU" in upper and/or lower case ASCII characters.

S-CODE - The S-CODE argument shall be one of the following single ASCII characters in upper or lower case. (RFC-765, pg 267)

F - File structure (no record structure)

R - Record Structure

COMMAND EXAMPLE - STRU F
- STRU R

2. The default file structure shall be File. (RFC-765, pg 267)

NOTE: See 3.6.6.2a and b for definition of File and Record structures.

e. TRANSFER MODE COMMAND (RFC-765, pg 267)

The transfer mode command shall specify the mode for file transfer.

1. COMMAND FORMAT - MODE[SP]M-CODE[byte-size][CRLF] (RFC-765, pg 285)

MODE - The command code shall be the character string "MODE" in upper and/or lower case ASCII characters.

M-CODE - The M-CODE argument shall one of the following single ASCII characters in upper or lower case. (RFC-765, pg 267)

S - Stream

B - Block

C - Compressed

L - Local Byte

byte-size - The number of bits associated with M-CODE = 'L'

COMMAND EXAMPLE - MODE S
- MODE B
- MODE C
- MODE L60

2. The default transfer mode shall be Stream. (RFC-765, pg 267)
NOTE: See 3.6.6.2a, b, and c for definition of Stream, Block, and Compressed modes.

f. SITE COMMAND (RFC-765, pg 272)

The SITE command allows an FTP user to specify typical NOS defaults (for example, alternate packnam, character set) so that they do not have to be entered for each command. It also allows a CYBER-to-CYBER transfer to be specified so that NOS-specific information is not lost. The format of the SITE command is:

SITE[SP]NOS[SP]p1,p2,p3,...,pn[CRLF] (RFC-765, pg 285)

where pn are order-independent parameters. The NOS keyword is required. If it is omitted, FTP will reject the SITE command. The following parameters are defined:

Parameter	Description	Affected FTP Command(s)
PN=packname	Pack name	all, incl. LIST
R=r	Device type	all
UN=username	User name	all, incl. LIST
RT=rt	CRM record type	all
MRL=mrl	CRM maximum record length	all
FL=fl	CRM fixed length	all
DA	Direct access	all
IA	Indirect access	all
LO	Local access	all (local FTP only)
PF	Permanent access	all
READEOI	Read-to-EOI flag	RETR only
CS=cs	Character set	all
TRUNC	Pad processing	STOR/APPE
PEER	CYBER-to-CYBER defaults	STOR/RETR/APPE

Once a parameter is specified in the SITE command, any subsequent command to which that parameter might apply will use the value of the parameter unless a different value is specifically given. A subsequent SITE command can change the defaults again, or a re-initialize (REIN) command will clear the effects of all SITE command parameters. Only parameters specified in the command are changed; other parameters retain their current default values.

The PEER parameter is a special parameter that informs FTP that its peer is another CYBER. This causes the following things to happen:

1) The representation type is changed to TYPE L 60 (local byte 60)

2) The file structure is changed to STRU R (record structure)

3) Subsequent transfers will interpret the FTP record/file mark codes differently so that NOS EOF marks will be retained through the transfer.

4) The READEOI flag is cleared if previously set.

Since the FTP codes are interpreted differently, it is important not to send a SITE NOS PEER command to a CYBER FTP if the other FTP is not a CYBER FTP. Doing so will result in a protocol error declared by one or the other FTP.

A local CYBER FTP user can send the SITE NOS PEER command to either the local or remote FTP. If sent to the remote FTP, the local FTP will see it and if a successful response is received from the remote FTP then the local FTP will change its status also. If sent to the local FTP, the local FTP will automatically send one to the remote FTP and if a successful response is received from the remote FTP then the local FTP will change its status and return a success response to the user. If the remote FTP returns an unsuccessful response, the local FTP will not change its status. It is therefore not possible for a local CYBER user to send a SITE NOS PEER command to a non-CYBER FTP, since presumably the non-CYBER FTP would reject the SITE command. However, a local non-CYBER FTP user could presumably send a SITE NOS PEER command to a remote CYBER FTP, causing the problem discussed earlier.

3.6.2.1.1.3 FTP Service Commands (RFC-765, pg 267)

The FTP service commands shall define the file transfer or the file system function requested by the user. The only order dependency allowed for FTP SERVICE COMMANDS shall be that RENAME FROM shall be followed by RENAME TO and that RESTART shall be followed with the interrupted FTP service command, such as RETRIEVE or STORE.

NOTE: In the COMMAND FORMAT subsections a.1 through j.1, items enclosed in brackets [] have the following meaning:

[SP] - represents one ASCII space.

[...] - represents a command argument.

[CRLF] - represents the TELNET end-of-line character string of Carriage Return Line Feed.

a. RETRIEVE COMMAND (RFC-765, pg 268)

This command shall cause FTP to transfer a copy of the named file to a server or user FTP at the other end of the of the data connection.

1. COMMAND FORMAT - RETR[SP][PATHNAME][CRLF] (RFC-765, pg 285)

RETR - The command code shall be the character string "RETR" in upper and/or lower case ASCII characters.

PATHNAME - The PATHNAME argument shall be as defined in section 3.6.2.5.1.

EXAMPLE - RETR FILE1,PN=ALTPACK,PW=PASSWRD

2. The status and contents of the file at the server FTP shall be unaffected by execution of the RETRIEVE COMMAND. (RFC-765, pg 268)

b. STORE COMMAND (RFC-765, pg 268)

This command shall cause FTP to accept the data from the data connection and to store the data as a NOS file.

1. COMMAND FORMAT - STOR[SP][PATHNAME][CRLF] (RFC-765, pg 285)

   STOR - The command code shall be the character string "STOR" in upper and/or lower case ASCII characters.

   PATHNAME - The PATHNAME argument shall be as defined in section 3.6.2.5.1.

   EXAMPLE - STOR FILE1,R=DQ,CT=PU,M=R

2. The server FTP shall create a file if the named file does not exist at the server site and shall replace the file with the new data if the named file does exist. The default file type shall be direct access. The SITE command shall be used to change the default file type to indirect access. (RFC-765, pg 268)

c. APPEND COMMAND (RFC-765, pg 268)

This command shall cause the server FTP to accept the data from the data connection and add it to the end of an existing file with the same name or create a new file to contain the data.

1. COMMAND FORMAT - APPE[SP][PATHNAME][CRLF] (RFC-765, pg 285)

   APPE - The command code shall be the character string "APPE" in upper and/or lower case ASCII characters.

PATHNAME - The PATHNAME argument shall be as defined in section 3.6.2.5.1.

EXAMPLE - APPE FILE1

d. RESTART COMMAND (RFC-765, pg 270)

The server FTP shall position the file to the marker and shall resume the file transfer only upon receipt of the appropriate service command from the user FTP. (RFC-765, pg 270)

1. COMMAND FORMAT - REST[SP][MARKER][CRLF] (RFC-765, pg 285)

   REST - The command code shall be the character string "REST" in upper and/or lower case ASCII characters.

2. MARKER - The argument field shall represent the server FTP marker at which the server FTP shall restart the file transfer. (RFC-765, pg 270)

3. The marker shall consist of a string of ASCII printable character codes of 33 through 126 decimal contained in a block header code of a file transferred in BLOCK or COMPRESSED mode. (RFC-765, pg 261,285,286.) Section 3.6.2.4.4 defines the NOS marker format.

   EXAMPLE - REST mrkrid

e. RENAME FROM COMMAND (RFC-765, pg 270)

This command shall specify the file to be renamed by the server FTP upon receipt of the RENAME TO command. Note: It is the responsibility of the user to issue the RENAME TO command.

1. COMMAND FORMAT - RNFR[SP][PATHNAME][CRLF] (RFC-765, pg 285)

   RNFR - The command code shall be the character string "RNFR" in upper and/or lower case ASCII characters.

   PATHNAME - The PATHNAME argument shall be as defined in section 3.6.2.5.1.

   EXAMPLE - RNFR FILE1,PW=PASSWRD

f. RENAME TO COMMAND (RFC-765, pg 270)

   This command shall specify the new file name for the file named in the immediately preceeding RENAME FROM command.

   1. COMMAND FORMAT - RNTO[SP][PATHNAME][CRLF] (RFC-765, pg 285)

      RNTO - The command code shall be the character string "RNTO" in upper and/or lower case ASCII characters.

      PATHNAME - The PATHNAME argument shall be as defined in section 3.6.2.5.1.

      EXAMPLE - RNTO FILE1

g. DELETE COMMAND (RFC-795, pg 271)

   This command shall cause the server FTP to delete the named file at the server site. A cautionary message shall be provided by user FTP to NOS interactive terminal users prior to file deletion.

1. COMMAND FORMAT - DELE[SP][PATHNAME][CRLF] (RFC-765, pg 285)

   DELE - The command code shall be the character string "DELE" in upper and/or lower case ASCII characters.

   PATHNAME - The PATHNAME argument shall be as defined in section 3.6.2.5.1.

   EXAMPLE - DELE FILE1

h. LIST COMMAND (RFC-765, pg 271)

This command shall cause the server FTP to send a list of files to the user FTP.

1. COMMAND FORMAT - LIST[SP][USERNUM][CRLF] (RFC-765, pg 285)

   LIST - The command code shall be the character string "LIST" in upper and/or lower case ASCII characters.

   USERNUM - The USERNUM argument shall be a NOS user number that shall consist of one to seven alphanumeric characters.

   It shall be an optional argument and, if used, shall specify an alternate NOS user catalog for which a list of files shall be produced.

   If the argument is not specified a list of files in the current catalog of the user shall be produced.

   EXAMPLES - LIST
   - LIST DDNUSER

i. COMMAND FORMAT - HELP[SP][KEY WORD][CRLF] (RFC-821, p29)

HELP - the command code shall be the character string "HELP" in upper and/or lower case ASCII characters. (RFC-821, pp25,27)

KEY WORD - The KEY WORD argument shall be an ASCII upper and/or lower case character string that indicates the type of help wanted.

EXAMPLE - HELP SITE

The topics to be supplied include a general help discussion (no topic name), a topic called TOPICS which list all possible help topics, PARAMETERS which lists all possible parameters, REMOTEUSER which provides remote user-specific info, LOCALUSER which provides local user-specific info, ERRORS which provides general error help, a set of Exxxx topics for specific errors where xxxx is a specific error code (the text will discuss possible corrections, not just reiterate the original error message text), EXAMPLE which gives an example of a file/mail transfer, and MINIMAL which describes the minimal set of commands and parameters to do a rudimentary file/mail transfer. There will also be at least one topic for each command and each parameter, plus topics on general subjects such as record structures, character sets, examples, etc.

j. NOOP COMMAND (RFC-765, pg 273)

This command shall specify that no action be taken by the server FTP except to send an OK reply.

1. COMMAND FORMAT - NOOP[CRLF] (RFC-765, pg 285)

NOOP - The command code shall be the character string "NOOP" in upper and/or lower case ASCII characters.

EXAMPLE - NOOP

k. ABORT COMMAND (RFC-765, pg 270)

This command shall cause FTP to abort the previous FTP service command and any associated ongoing transfer of data.

1. COMMAND FORMAT - ABOR[CRLF] (RFC-765, pg 285)

ABOR - The command shall be the character string "ABOR" in upper and/or lower case ASCII characters.

EXAMPLE - ABOR

3.6.2.2 FTP Replies (RFC-765, pg 274)

Replies to FTP commands shall be used to ensure synchronization of requests and actions during the process of file transfer, and to guarantee that the user process always knows the state of the server.

NOTE: In the following paragraphs [SP] means an ASCII space and [CRLF] means a TELNET end of line (ASCII Carriage Return followed by an ASCII Line feed).

a. At least one reply shall be generated by the FTP server for every FTP command received. (RFC-765, pg,274)

b. A single line FTP reply shall consist of a 3-digit code transmitted as three alphanumeric characters followed by a [SP], followed by one line of text, followed by a [CRLF]. (RFC-765, pg,274)

c. Multi line replies shall consist of a 3-digit code, followed immediately by a Hyphen "-" (also known as a minus sign), followed by text. The last line shall begin with the same 3-digit code, followed by a [SP], some optional text, and terminated by a [CRLF]. (RFC-765, pg 275)

3.6.2.2.1 FTP Reply Codes (RFC-765, pg 276)

The reply codes and their meaning shall be as shown below:

a. The first digit, most significant, of the reply code shall have one of the following values: (RFC-765, pg 276)

1. 1xx - Positive preliminary reply (RFC-765, pg 276)

   The requested action is being initiated; expect another reply before proceeding with a new command. The server FTP shall queue additional commands received while the preceding command is in progress.

2. 2xx - Positive completion reply (RFC-765, pg 276)

   The requested action has been successfully completed and the user FTP may initiate a new request.

3. 3xx - Positive intermediate reply (RFC-765, pg 276)

   The command has been accepted. Action is held in abeyance pending receipt of another command from the user FTP that specifies required information.

4. 4xx - Transient negative reply (RFC-765, pg 276)

   The command has not been accepted and requested action did not take place. The error condition is temporary and the user FTP should reissue the command or return to the beginning of the command sequence.

5. 5xx - Permanent negative completion reply (RFC-765, pg 277)

   The command has not been accepted and the requested action did not take place. The user FTP should not repeat the exact command or command sequence.

b. The second digit of the reply code shall indicate functional grouping with one of the following values: (RFC-765, pg 277)

   1. x0x - Syntax error (RFC-765, pg 277)

      These replies refer to syntactically correct commands that don't fit any functional category, or are unimplemented or superfluous.

   2. x1x - Information (RFC-765, pg 277)

      These replies are for requests for information such as status or help.

   3. x2x - Connections (RFC-765, pg 277)

      These replies refer to the TELNET and data connections.

   4. x3x - Authentication and accounting (RFC-765, pg 277)

      These replies are for the login process and accounting.

   5. x4x - Unspecified (RFC-765, pg 277)

   6. x5x - File system (RFC-765, pg 278)

      These replies indicate the status of the server FTP file system based upon the requested transfer or other file system action requested by the user FTP.

c. The third digit, least significant, shall give a more precise meaning to each of the function categories specified by the first and second digit defined in above subsections a. and b. (RFC-765, pg 278)

The text associated with the following listed replies is only a recommendation; however, the first and second digit shown are mandatory and shall not be changed. (RFC-765, pg 278)

The third digit of the reply codes shall be used with the first and second digit of the reply code functional groups as follows: (RFC-765, pg 278)

1. 200 Command OK (RFC-765, pg 278)

2. 500 Syntax error, command unrecognized. (RFC-765, pg 278) (The above may include errors such as command line too long.)

3. 501 Syntax error in parameters or arguments. (RFC-765, pg 278)

4. 202 Command not implemented, superfluous at this site. (RFC-765, pg 278)

5. 502 Command not implemented. (RFC-765, pg 278)

6. 503 Bad sequence of commands. (RFC-765, pg 278)

7. 504 Command not implemented for a parameter. (RFC-765, pg 278)

8. 110 Restart marker reply. (RFC-765, pg 278)
   The text of the above shall be exactly as shown below:

   MARK yyyy[SP]=[SP]mmmm
   where yyyy is the user FTP data stream marker and
   mmmm is the server FTP equivalent marker.

9. 119 Terminal not available (RFC-765, pg 279)
   The above is not applicable per SOW.

10. 211 System status or system help reply. (RFC-765, pg 279)

11. 212 Directory status. (RFC-765, pg 279)

12. 213 File status. (RFC-765, pg 279)

13. 214 Help message. (RFC-765, pg 279)
    (Text on how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.)

14. 215 [scheme] is preferred scheme. (RFC-765, pg 279)
    The above is not applicable per SOW.

15. 120 Not implemented.

16. 220 Service ready for new user. (RFC-765, pg 279)

17. 221 Service closing TELNET connection. (RFC-765, pg 279)

18. 421 Service not available, closing TELNET connection. (RFC-765, pg 279)
    This may be a reply to any command if the FTP server knows it must shut down.

19. 125 Data connection already open; transfer starting (RFC-765, pg 279)

20. 225 Data connection open; no transfer in progress. (RFC-765, pg 279)

21. 425 Cannot open data connection. (RFC-765, pg 279)

22. 226 Closing data connection. (RFC-765, pg 279)
Requested file action successful (for example, file transfer or file abort).

23. 426 Connection closed; transfer aborted. (RFC-765, pg 279)

24. 227 Entering Passive Mode. (RFC-765, pg 279)

25. 230 User logged in, proceed. (RFC-765, pg 279)

26. 530 Not logged in. (RFC-765, pg 279)

27. 331 User name okay, need password. (RFC-765, pg 279)

28. 332 Need account for login. (RFC-765, pg 279)

29. 532 Need account for storing files. (RFC-765, pg 279)

30. 150 File status okay; about to open data connection. (RFC-765, pg 279)

31. 151 User not local. (RFC-765, pg 279)
The above is not applicable per SOW.

32. 152 User unknown; mail will be forwarded by the operator. (RFC-765, pg 279)
    The above is not applicable per SOW.

33. 250 Requested file action okay; completed. (RFC-765, pg 279)

34. 350 Requested file action pending further information. (RFC-765, pg 279)

35. 450 Requested file action not taken; file unavailable. (RFC-765, pg 279)
    (for example, file busy)

36. 550 Requested action not taken; file unavailable. (RFC-765, pg 279)
    (for example, file not found, no access)

37. 451 Requested action aborted; local error in process. (RFC-765, pg 280)

38. 551 Requested action aborted; page type unknown. (RFC-765, pg 280)

39. 452 Requested action not taken; insufficient storage space in system. (RFC-765, pg 280)

40. 552 Requested file action aborted; exceeded storage allocation (for current directory or dataset). (RFC-765, pg 280)

41. 553 Requested action not taken; file name not allowed. (RFC-765, pg 280)

42. 354 Start mail input. (RFC-765, pg 280)
The above is not applicable per SOW.

A numerically ordered list of reply codes can be found on page 280 of RFC-765.

3.6.2.2.2 Sequencing of Commands and Replies (RFC-765, pg 287)

The communication between the user FTP and the server FTP shall be an alternating dialog implemented as follows.

a. The FTP shall issue commands to the server FTP and shall wait for a reply before sending further commands. (RFC-765, pg 287)

b. The receiver FTP shall send one or more replies to the user FTP for each received command. (RFC-765, pg 287)

c. The specific COMMAND-REPLY sequences shall be as stated below. (RFC-765, pg 287)

NOTE: In the subsections d. through k. below each command is listed and is followed by possible reply codes. Preliminary replies are listed first with succeeding replies indented under them. Then positive and negative completion, and finally intermediary replies with the remaining commands from the sequence following.

d. Connection Establishment (RFC-765, pg 287)

120
  220
220
421

e. (This subparagraph not used)

f. Login (RFC-765, pg 288)

1. USER (RFC-765, pg 288)

230
530
500, 501, 421
331, 332

2. PASS (RFC-765, pg 288)

230
202
530
500, 501, 503, 421
332

3. ACCT (RFC-765, pg 288)

230
202
530
500, 501, 503, 421

g. Logout (RFC-765, pg 288)

1. QUIT (RFC-765, pg 288)

221
500

2. REIN (RFC-765, pg 288)

120
220
220
421
500, 502

h. Transfer parameters (RFC-765, pg 288)

1. PORT (RFC-765, pg 288)

200
500, 501, 421, 530

2. PASV (RFC-765, pg 288)

227
500, 501, 502, 421, 530

3. MODE, TYPE, and STRU (RFC-765, pg 288)

200
500, 501, 504, 421, 530

i. File action commands (RFC-765, pg 288)

1. REST (RFC-765, pg 288)

500, 501, 502, 421, 530
350

2. STOR (RFC-765, pg 289)

125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 452, 553
500, 501, 421, 530

3. RETR (RFC-765, pg 289)

125, 150
(110)
226, 250
425, 426, 451
450, 550
500, 501, 421, 530

4. LIST (RFC-765, pg 289)

125, 150
226, 250
425, 426, 451
450
500, 501, 502, 421, 530

5. APPE (RFC-765, pg 289)

125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 550, 452, 553
500, 501, 502, 421, 530

6. RNFR (RFC-765, pg 289)

   450, 550
   500, 501, 502, 421, 530
   350

7. RNTO (RFC-765, pg 289)

   250
   532, 553
   500, 501, 502, 503, 421, 530

8. DELE (RFC-765, pg 289)

   250
   450, 550
   500, 501, 502, 421, 530

9. ABOR (RFC-765, pg 290)

   250
   500, 501, 502, 421

   (Note that the RFC contains an apparent inconsistency. The command-reply sequence section for ABOR (RFC pg.48) indicates that 225 (data connection open; no transfer in progress) is an appropriate reply to ABOR. The ABOR command description (RFC pg.271) indicates that the data connection is always terminated. Since both are not possible, FTP will never respond to an ABOR command with a 225.)

j. Informational commands (RFC-765, pg 290)

1. HELP (RFC-765, pg 290)

211, 214
500, 501, 502, 421

k. Miscellaneous commands (RFC-765, pg 290)

1. NOOP (RFC-765, pg 291)

200
500, 421

3.6.2.3 Establishing Connections (RFC-765, pg 257, 282)

Before the transfer of data can take place, the user and server processes must establish a TELNET connection for exchanging FTP commands and replies and a data connection over which files may be transferred. The establishment of these connections shall be as stated below.

3.6.2.3.1 Establishing TELNET Connection (RFC-765, pg 282)

a. The user process shall initiate the full-duplex TELNET connection. (RFC-765, pg 282)

b. The server process shall "listen" on port L for initiation of the TELNET connection. (RFC-765, pg 282)

c. The TELNET connection shall be closed by the server process at the user's request after all transfers and replies have been completed. (RFC-765, pg 282)

d. The user and server processes shall follow the conventions of the TELNET protocol specified in section 3.5 of this document. (RFC-765, pg 282)

3.6.2.3.2 Establishing Data Connection (RFC-765, pg 257)

a. The user process default data port shall be the same as the the TELNET control connection port; that is, port U. (RFC-765, pg 257)

b. The server process default data port shall be adjacent to the TELNET control connection port, that is, port L-1. (RFC-765, pg 257)

c. The passive process (user or second server) shall "listen" on the data port prior to sending a transfer request command. (RFC-765, pg 257)

d. The server, upon receiving the transfer request, shall initiate the data port connection. (RFC-765, pg 257)

e. The server shall accept alternate data port commands. (RFC-765, pg 257)

f. The closing of the data port shall be by the server and shall be indicated to the user process by a reply. (RFC-765, pg 258)

g. The server shall unconditionally close the data port connection under the following conditions: (RFC-765, pg 258)

1. The transfer mode requires a close to indicate EOF. (RFC-765, pg 258)

2. An abort command is received from the user. (RFC-765, pg 258)

3. The port specification is changed by the user. (RFC-765, pg 258)

4. The TELNET connection is closed legally or otherwise. (RFC-765, pg 258)

5. An irrecoverable error condition occurs. (RFC-765, pg 258)

3.6.2.4 FTP Transmission Modes (RFC-765, pg 258)

The FTP shall provide three file transmission modes: one which formats data and allows for restart, one which compresses data for more efficient data transfer, and one which transfers the data with little or no processing.

3.6.2.4.1 FTP Stream Mode (RFC-765, pg 259)

a. The data shall be transferred as a stream of bytes. (RFC-765, pg 259)

b. Record structure shall be allowed. (RFC-765, pg 259)

c. For record structures, the EOR and EOF shall be indicated by a two-byte control code. The first byte shall be all ones, the escape character. The second byte shall be 1 for EOR, 2 for EOF, or 3 for EOR/EOF. Section 3.6.2.5.4 specifies the mapping FTP and NOS file structures. (RFC-765, pg 259)

d. For file structure, the EOF shall be indicated by closing the data port connection. (RFC-765, pg 259)

3.6.2.4.2 FTP Block Mode (RFC-765, pg 259)

a. The data shall be transferred as a series of blocks preceded by one or more header bytes. (RFC-765, pg 259)

b. The header byte shall consist of descriptor code and a count field. (RFC-765, pg 259)

1. The descriptor code (decimal) shall be as follows: (RFC-765, pg 260)

128 = End of data block is EOR
64 = End of data block is EOF
32 = Suspected errors in data block
16 = Data block is a restart marker

Section 3.6.2.5.4 specifies the mapping between FTP and NOS file structures.

2. The count field shall indicate the total length of the data data block. (RFC-765, pg 259)

3.6.2.4.3 FTP Compressed Mode (RFC-765, pg 261)

There are three kinds of information sent in compressed mode as stated below.

a. Regular data shall be sent in a byte string where the first byte contains a count of the number of data bytes that follow. The count shall be >0 and <=127 with the MSB always 0. (RFC-765, pg 261)

b. Compressed data shall consist of replication or filler bytes. Replicated data shall be indicated by a first byte with the two MSBs of the first byte set to 10 followed by a 6-bit replication count, followed by the repeated character. Filler bytes shall be indicated by the two MSBs of the first byte being set to 11 followed by 6-bit count. The actual filler byte shall be a space for ASCII and zero for Image data representation types. (RFC-765, pg 262)

c. Control information shall be sent in two-byte escape sequences and shall be as described in 3.6.2.4.2.b1. (RFC-765, pg 261)

3.6.2.4.4 Restart Markers

1. CYBER FTP will choose restart markers based upon the elapsed time since the last restart marker was transmitted. The elapsed time (in seconds) can be specified upon invocation of FTP, or if not specified then, by an installation selectable default time. The initial default will be 60 seconds. The FTP invocation parameter is RESTART=n where n is the number of seconds.

2. Restart markers will be based upon the current file, record, and byte number (bytes are 60 bits for TYPE L 60, 8 bits for all others). It will be of the following format:

```
      fffffffff,rrrrrrrrrr,bbbbbbbbbb              (format 1)
-or-  fffffffff,rrrrrrrrrr,aaaaa,bbbbbbbbbb       (format 2)
```

where:

fffffffff up to 10 digit decimal number identifying the file (NOS logical file) number, counting from file 1.

rrrrrrrrrr up to 10 digit decimal number identifying the record (NOS logical record) number, counting from record 1.

bbbbbbbbbb up to 10 digit decimal number identifying the byte number at which to begin restart. If the byte count is larger than 10 digits then format 2 is used (see aaaaa parameter).

aaaaa — up to 5 digit decimal number identifying the high-order byte number. This number is prefixed to the bbbbbbbbbb parameter to obtain the total byte count. This parameter is present only if the byte count exceeds 10 digits. Note that the total byte count cannot exceed 2**48-1 (281474976710655).

For all numbers, FTP suppresses leading zeros during marker generation and ignores any leading zeros supplied in the REST command.

3. Example (format 1): If FTP chooses a restart marker in file 1, record 10, byte 60 then FTP would send a marker (assume block mode) of the form:

Descriptor Code = 16	Byte Count = 7		Marker Char '1'	Marker Char ','	Marker Char '1'
Marker Char '0'	Marker Char ','	Marker Char '6'	Marker Char '0'		

The restart reply issued to the user by the remote FTP (assume non-CYBER) would be:

```
110 MARK ?????? = 1,10,60
```

where ?????? is the equivalent marker for the remote FTP and is host dependent.

4. Example (format 2): If FTP chooses a restart marker in file 356, record 2, byte 181474976710655 then FTP would send a marker (assume block mode) of the form:

Descriptor Code = 16	Byte Count = 22		Marker Char '3'	Marker Char '5'	Marker Char '6'
Marker Char ','	Marker Char '2'	Marker Char ','	Marker Char '1'	Marker Char '8'	Marker Char '1'
Marker Char '4'	Marker Char '7'	Marker Char ','	Marker Char '4'	Marker Char '9'	Marker Char '7'
Marker Char '6'	Marker Char '7'	Marker Char '1'	Marker Char '0'	Marker Char '6'	Marker Char '5'
Marker Char '5'					

The restart reply issued to the user by the remote FTP (assume non-CYBER) would be:

```
110 MARK ?????? = 356,2,18147,4976710655
```

where ?????? is the equivalent marker for the remote FTP and is host dependent.

3.6.2.5 CYBER/NOS Unique Features

The following subsections define the FTP command parameters and features that are unique to the CYBER NOS implementation of FTP.

3.6.2.5.1 NOS Pathname Parameters

A NOS pathname consists of a file name plus any legal permanent file access commands and other parameters. Some NOS parameters are allowed in FTP commands while other parameters are not.

a. The following NOS parameters are permitted:

Parameter	Description	Applicable FTP Command(s)
AC=ac	Alternate catlist access	STOR only
AL=level	Access level	STOR only
BR=br	Backup requirement	STOR only
CT=ct	File permit category	STOR only
M=m	File access mode	STOR/RETR only
PN=packname	Pack name	all, incl. LIST
PR=pr	Preferred residence	STOR only
PW=password	File password	all, xcept SITE
R=r	Device type	all
S=space	Minimum available space	STOR only
SS=subsystm	Interactive subsystem	STOR only
UN=username	User name	all, incl. LIST
XD=expdate	Password expiration date	STOR only
XT=exptime	Password expiration time	STOR only

b. The following NOS parameters are not permitted:

Parameter	Description
NA	No abort
WB	Wait-if-busy
RT	Real-time processing

c. The following FTP-unique parameters are permitted:

Parameter	Description	Applicable FTP Command(s)
RT=rt	CRM record type	all
MRL=mrl	CRM maximum record length	all
FL=fl	CRM fixed length	all
DA	Direct access	all
IA	Indirect access	all
LO	Local access	all (local FTP only)
PF	Permanent access	all
FSC=n	File skip count	RETR only
READEOI	Read-to-EOI flag	RETR/SITE only
CS=cs	Character set	all
TRUNC	Pad processing	STOR/APPE/SITE
PEER	CYBER to CYBER defaults	SITE only

(Note that these parameters are meaningless for the DELE, RNFR and RNTO commands.)

d. Many of the parameters listed above are meaningful only for direct access files. If a direct access file parameter is supplied (for example, "M") and the file is actually indirect access, the parameter is ignored and no warning is issued.

e. The usage of the parameters listed in paragraph a, above, is defined in the NOS Reference Manual, Vol 3. Explanations of the FTP-unique parameters in paragraph c are in subsections 3.6.2.5.2 through 3.6.2.5.7, below.

3.6.2.5.2 CRM Parameters

CRM parameters are used to specify the desired record type to be used in CYBER FTP processing.

a. The following parameters are permitted in the pathname and SITE command to specify CRM information:

Parameter	Description
RT=n	CRM record type; values are W, Z, or S, see table below for defaults
MRL=nnn	CRM maximum record length; optional for STOR (write) on W records, optional for RETR (read) on W and S records, not used for all others, no default
FL=nnn	CRM full length; optional for STOR/RETR on Z records, not used for all others, default is installation selectable but typically 150 characters

b. FTP will support three CRM record types: W (word-count), Z (zero-byte terminated), and S (NOS system logical record). The following table shows which records are the default, and which are legal, for each FTP representation type:

FTP type	W	Z	S
ASCII (6/12)	X	D	X
ASCII (8/12)	D	-	X*
Image	D	-	X*
Logical 60	-	-	D

Key: X Combination is permitted
D Default CRM record for given FTP type
* File may not duplicate for this combination
- Combination is not permitted

1. S-type records

CRM S-type records correspond to NOS system logical records. No special data organization is assumed; the data is treated as a stream of 60-bit words. This can cause problems with files exchanged with foreign host systems (see item b. below). An advantage of this record type is that most system utilities will operate correctly with this type but not with W-type records. S-type records should be chosen when system utilities must be used to interpret the data (that is, utilities which don't support W-type records) and when the end of data can be distinguished from trailing zero filler without the need for a byte count.

2. W-type records

CRM W-type records maintain a header that contains a character count. The CRM user never sees this header; it is stripped off by CRM before data is given to the user. An FTN program using standard I/O can take advantage of this also by using the FILE control statement to specify the W-type. Normal system routines, such as FSE, will incorrectly interpret this header as data. If system routines must be used, FORM can be used to convert between record types (but then duplication problems, such as described in item b. below, may occur). W-type records should be chosen when only user programs will be used to interpret the data, and when the end of data can only be known by the byte count.

The character count returned by CRM W-type records is in 6-bit characters. To convert from 6-bit to 8-bit characters, use this formula:

$$i = INT(6j/8)$$

where: i = # of 8-bit characters
j = # of 6-bit characters
INT = integer truncation function

To convert from 8-bit to 6-bit characters, use this formula:

$$j = INT(8i/6)$$

3. Z-type records

CRM Z-type records are used to delimit lines of extended disply code text, where each line ends with a zero-byte terminator. All system utilities that do text processing (for example, FSE) use this record format. Z-type records should be chosen for text files, particularly when the files will be interpreted by system utilities.

c. Two type/record combinations may not permit exact duplication. For example, if the file is sent to a CYBER from an 8-bit machine, sent back to the 8-bit machine, and then the original and transferred files compared, they may not be identical.

Failure to duplicate occurs for an image or ASCII 8/12 representation type with a S-type CRM record. In this case, when CYBER FTP sends a file out, it interprets all trailing 8-bit bytes with a value of zero as padding, and does not send them. If a file was originally sent to the CYBER with trailing zero-bytes, these will not be returned (since they are interpreted as padding). Since S-type records have no way of identifying padding, this cannot be resolved. The default for image type files is a W-record, so if the user selects S-type records, the user must have some way of determining the end of a file, or must know that no trailing zero-bytes exist, or must not care.

ASCII 6/12 files do not present a problem, since no character is represented as a zero-byte, and trailing zero-bytes can be freely interpreted as padding.

3.6.2.5.3 NOS File Parameters

NOS file parameters are used to specify the CYBER file type to be used in FTP processing.

a. NOS defines two types of permanent files, direct and indirect access files. Four keyword parameters have been defined in FTP to support these file types plus local files for local FTP users. The parameter can be supplied as a pathname parameter or in the SITE command. The four keywords are PF, IA, DA, and LO, with no values. If IA is supplied, then the file will be read/written as an indirect access file. If DA is supplied, the file will be read/written as a direct access file. If LO is supplied, the file will be

read/written as a local file. If PF is supplied, the file will be read/written as a permanent (indirect or direct access) file. The four parameters are mutually exlusive, resulting in a syntax error if more than one is supplied in the same command.

b. If on a command, one of the keywords is supplied and the actual file type does not match, an error will be returned and the command will fail.

c. If no keyword is specified, the action depends on whether the file already exists and on the current default file type. The normal default file type is installation selectable but is typically indirect access. The default can be changed by the SITE command. A default type of LO is always ignored (for both a read (RETR) and a write (STOR,APPE)) if the request comes from a remote user. A default type of IA or DA is always ignored for read requests from both local and remote users. A default type of IA or DA is ignored for write requests from both local and remote users if the file already exists, in which case the current file type is used; if the file doesn't exist, then it is created with the default IA or DA file type. A default of LO for both local read and local write requests causes FTP to search for (or create) a local file. The PF parameter is primarily used to override a default of LO without having to specify IA or DA, and so is handled in the same way as IA/DA.

d. When the default is ignored on a read, the file will be ATTACHed or GETted depending on the existing file type. Therefore, it is generally unnecessary in any command to specify a file type unless the user specifically requires a certain type. The following table

summarizes the different default type, user type, and operation combinations:

Request Source	Operation: Default Type	Read (RETR) File Exists	Write (STOR,APPE) File Exists	Write (STOR,APPE) No File Exists
Local	Indirect IA	Actual	Actual	Indirect
	Direct DA	Actual	Actual	Direct
	Local LO	Local	Local	Local
	Permanent PF	Actual	Actual	Indirect
Remote	Indirect IA	Actual	Actual	Indirect
	Direct DA	Actual	Actual	Direct
	Local LO	Actual	Actual	Indirect
	Permanent PF	Actual	Actual	Indirect

3.6.2.5.4 EOR/EOF Processing Parameters

The EOR/EOF parameters allow FTP transfer of NOS multi-file files using FTP.

a. NOS defines three types of file marks: EOR (end-of-record), EOF (end-of-file) and EOI (end-of-information). FTP only defines two: EOR (end-of-record) and EOF (end-of-file). The NOS and FTP EOR marks correspond, while the NOS EOI corresponds to the FTP EOF. There is no FTP equivalent to the NOS EOF.

b. NOS FTP solves this problem by normally treating a NOS EOF as an FTP EOF. Since FTP will stop transferring a file when it receives an EOF mark, this implies that only one NOS file in a multi-file file will be transferred. Two parameters are supplied which extend these capabilities:

FSC=n File skip count. The default is zero (that is, the first file of a multi-file file will be transferred). If supplied, n NOS file marks will be skipped before transfer begins (that is, file n+1 of the multi-file file will be transferred). This parameter is meaningless for any command except RETR (or the source file of a SEND/RECV command) and is ignored.

READEOI Read-to-EOI flag. If supplied, all files of the multi-file set will be transferred, with NOS EOF marks converted to FTP EOR marks. This implies that the file cannot be duplicated if re-transferred to the CYBER; that is, the NOS EOF marks cannot be re-constructed. This parameter is meaningless for any command except RETR (or the source file of a SEND/RECV command) and is ignored. This parameter allows a user to transfer an entire multi-file file if NOS EOF marks are not important.

c. CYBER-to-CYBER transfers can be done which preserve the NOS EOR/EOF/EOI structure using the PEER parameter on the SITE command (see SITE command). In this case, since FTP knows that the peer is another CYBER, the codes are interpreted differently so the standard FTP protocol can be used to identify NOS EOF marks. In this case, the FTP EOF code (2) is interpreted as a NOS EOF, the FTP EOF/EOR code (3) is interpreted as a NOS EOI, and a NOS EOF/EOI is represneted as two separate FTP marks.

3.6.2.5.5 ASCII Character Set Parameter

The character set parameter allows selection of the ASCII code representation to be used during FTP processing.

a. FTP supports two character sets: ASCII 6/12 (extended display code) and ASCII 8/12 (ASCII characters right-justified in 12 bits). The default character set is ASCII 6/12. If 6/12 is selected, the characters are converted between 6/12 and ASCII 8/8 using standard conversion tables from FCOPY. If Z-type records are selected, end-of-lines (CRLF) are converted to zero-byte terminators; otherwise conversion is character-for-character with the full ASCII character set supported. Note that the only end-of-line sequence supported for conversion to Z records is CRLF. If another line terminator is employed (for example, US or LFCR) then the file must be transferred using 8/12 mode and converted using FCOPY.

b. The character set can be specified via a parameter in the pathname or SITE command:

CS=n Character set. n = "ASCII", "A", "ASCII8", or "8". If ASCII or A are supplied then ASCII 6/12 is assumed. If ASCII8 or 8 are supplied then ASCII 8/12 is assumed. The parameters correspond to FSE parameters. The default local character set, if no character set is specified, is installation selectable but is typcially ASCII 6/12. This default can be changed by the SITE command.

3.2.5.2.6 TRUNCate File Parameter

a. The parameter TRUNC is defined for both the pathname and the SITE command.

b. Although CRM is sufficient to handle an incoming file from a foreign host which is then retransmitted to check duplication, it is not sufficient to handle a native CYBER file transmitted to a foreign host and then retransmitted to the CYBER. The TRUNC parameter is intended primarily to handle the case where an image file is being sent to a CYBER, and it is known that the file originally resided on a CYBER. For example, a file is transferred from a CYBER to an 8-bit machine and then from the 8-bit machine to a CYBER. Suppose the file had only one 60-bit word. In the CYBER-to-remote transfer, an extra 4 bits of padding was inevitably added to the file to make it a multiple of 8-bit bytes. When the file is transferred to the CYBER, FTP will not know that the 4 bits are pad, and will allocate another word with another 56 bits of padding. In this case, the user should specify TRUNC, the 4 bits of padding will be dropped, and the file will be an exact duplicate of the original.

c. If the TRUNC parameter is not specified and the default has not been changed by a SITE command then the word is filled out with 56 zero bits if a byte overflows into the next CYBER word. Note that TRUNC only applies to a 4-bit overflow. If one full byte or more is stored into the last CYBER word, the word is always zero-filled regardless of whether or not TRUNC is specified. Note that pad processing applies to every record in a file, not just the end of the file.

d. If it is possible for zero-fill within a file to be confused with file data, or if the file is to be sent out again, a CRM record type must be chosen that specifically identifies the padding bytes (see section 3.6.2.5.2). Otherwise, when the file is sent to another host, the zero-fill will be sent as data, or a program reading the file may interpret the zero-fill as data.

e. The TRUNC parameter is not required for TYPE L 60 transfers or SITE NOS PEER transfers, since TRUNC will automatically be assumed. It is also generally not required for ASCII files, unless the trailing character(s) of records in the file can be mistaken for zero-fill padding. For 6/12 ASCII files, this is not possible, since no character uses the zero-byte for its representation.

3.6.2.5.7 CYBER-to-CYBER Transfers

This section describes FTP methods for facilitating CYBER-to-CYBER file transfers.

a. In addition to the ASCII (A) and Image (I) representation types, FTP supports a Local Byte type. The form of the command is:

TYPE L 60

b. Note that only a value of 60 is supported; other values will be rejected. If this type is specified, FTP will assumed that all data received will be a multiple of 60 bits. For example, if 8 bytes (8-bit bytes) are received for a complete transmission, the last 4 bits will be discarded. Normally, using other representation types, a second word would be created containing the 4 bits and 56 bits of zero padding. This parameter was added to make it easier to send files CYBER-to-CYBER and ensure that the files are exact duplicates. In this mode, only system logical records (S-type) can be selected, but since the files are exact duplicates, any type file will be successfully moved, including W-type, indexed sequential, etc. Therefore, if the RT parameter is specified other than S with an L 60 type transfer, it will be ignored with a warning message. It is recommended that the SITE NOS PEER command be issued to ensure identical transfer of file mark information also. (Reference section 3.6.2.1.1, SITE Command.)

3.6.3 External Interfaces (RFC-765)

The following subsections define the FTP external interfaces. Interfaces are provided for direct user call of FTP and for application calls to FTP.

3.6.3.1 Application Program (RFC-765)

The Application Program interface shall be provided by FTP to application programs written in FORTRAN or any language that can interface to FORTRAN (for example, SYMPL and COBOL). The FTP Control Statement interface shall use this interface to access the FTP capabilities. In the following discussions, the term 'standard_character' string is used to refer to a FORTRAN character string or to an array of 60-bit words that contains left-justified characters, zero-filled and zero-byte terminated. A standard_character string uses the 6/12 display code representation, so the display code escape characters (caret and at-sign) will always be interpreted as such. If a standard_character string can contain n characters, then the

calling routine should allocate 2n 6-bit bytes (10 bytes per 60-bit word) for the worst case of all lower case characters. The term 'name_character' is used to refer to the character subset consisting of alphabetic (A-Z), numeric (0-9), minus sign (-), and period (.) characters.

3.6.3.1.1 FTP_Accept_Reply (FTPAR) Routine (RFC-765)

The FTP_Accept_Reply routine shall return the full text-response to the last request issued. This routine can also be used to wait for completion of requests that were previously issued with no_wait. The calling format shall be:

```
CALL FTPAR  (connection_id, no_wait, last_command, reply, reply_code,
             more_reply)
```

connection_id — FTP connection identifier. Integer value supplied by the application. This must be the identifier returned by FTP in the FTP_Connect request. The application uses this value to distinguish between different FTP connections simultaneously established.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately if the previous request has not completed. The application must then call FTP_Accept_Reply again later to determine the reply.

last_command — Last command issued by the application. A standard_character string returned by FTP containing the 4-character command; parameters originally supplied with command are not returned.

reply — Textual reply from connection attempt. A standard_character string returned by FTP containing up to 80 characters of reply text. The text for replies from a CYBER FTP, either local or remote, is specified in section 3.6.2.2.1. Reply text from other FTP implementations is defined by that implementation. This value is undefined if reply_code is zero or negative.

reply_code — Reply code from previous request. Integer value is returned by FTP for completed requests regardless of success. Reply codes are defined in section 3.6.2.2.1. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

more_reply — Indicates the reply text is continued on another line (80 character array). Boolean value returned by FTP. If true, a subsequent call to FTP_Accept_Reply will return the next line of the message. When the last line is read (more_reply false), a subsequent call to FTP_Accept_Reply will return the first line of the response.

3.6.3.1.2 FTP_Connect (FTPC) Routine (RFC-765)

The FTPC routine is called to begin an FTP session. All remote hosts involved in the session (a maximum of two, for third-party transfers) must be specified in the single call. The entire FTP session can be terminated by calling FTPD, or FTPC can be called again to specify a new session. The format of the FTPC command is:

```
CALL FTPC (host_address, host2_address, no_wait,
           connect_id, connect2_id, reply_code)
```

host_address — Address of remote host. A standard character string supplied by the application containing up to 24 name characters. The name can be an internet address or a domain name. If a domain name is supplied, FTP will determine the internet address. If the string is empty or blank, the local host is assumed. If an internet address is supplied, it must be delimited by brackets (for example, [10.0.3.19]).

host2_address — Address of second remote host. The same rules apply as above, if this parameter is used. This should be zero if the transfer is between the local host and a remote host specified by host_address. The host2_address should be the address of the second host if the transfer is between two remote hosts; that is, a third-party transfer.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call FTP_Accept_Reply to subsequently determine the reply.

connection_id — Remote host FTP connection identifier. Integer value is returned by FTP for successful connections and must be supplied on subsequent FTP calls to identify the specific connection. The connection id for the local host is always 1. A connection to the local host is always assumed. However, in a third-party session, no file transfer commands can be directed to the local host. Only the HELP, QUIT, REIN, and NOOP commands can be sent to the local host during third-party sessions.

connect2_id — Undefined if a zero host2_address is specified. Otherwise it contains an integer value, as above, used to identify the third-party connection.

reply_code — Reply from connection attempt. Integer value is returned by FTP for all connection attempts regardless of success. Reply codes are defined in section 3.6.2.2.1. If the application requires the message text, the FTP_Accept_Reply routine must be called. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

Any commands directed at remote hosts are checked by FTPI, as far as possible, for correct syntax and for correct entry given the current state. FTPI advances the state depending on responses from the commands. Unsupported commands (that is, specified in FTP RFC, but not supported by CYBER implementation) are passed unexamined with no state change. Unspecified commands (commands not in RFC) are rejected by FTPI as erroneous. The states of the two transfer connections are both checked. For example, a RETR command cannot be sent to a remote FTP (in a two-party session) if the local host has not first been sent a STOR command. The SITE command is given special processing as defined in section 3.6.2.1.1.2f.

3.6.3.1.3 FTP_Disconnect (FTPD) Routine (RFC-765)

The FTP_Disconnect routine shall request the FTP Protocol Interpreter to terminate a the established connections. The calling format shall be:

CALL FTPD (no_wait, reply_code)

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call FTP_Accept_Reply to subsequently determine the reply.

reply_code — Reply from disconnect attempt. Integer value is returned by FTP for all attempts regardless of success. Reply codes are defined in section 3.6.2.2.1. If the application requires the message text, the FTP_Accept_Reply routine must be called. Reply code is zero if no_wait is specified and operation is currently in progress.

3.6.3.1.4 FTP_Status (FTPS) Routine

The FTP_Status routine shall return the current status of an FTP connection. The calling format shall be:

CALL FTPS (connection_id, in_progress, host_address, user_name, account, data_type, file_structure, transfer_mode, pathname, reply_code)

connection_id — FTP connection identifier. Integer value supplied by the application. This must be the identifier returned by FTP in the FTP_Connect request. The application uses this value to distinguish between different FTP connections simultaneously established.

in_progress — Indicates if a command is currently in progress. Boolean value returned by FTP is true if the user has issued a command that has not yet completed.

host_address — Address of remote host. A standard_character string returned by FTP containing up to 24 name_characters. This is the name supplied to FTP in the FTP_Connect request.

user_name	Name of user. A standard_character string returned by FTP containing up to 80 characters. This is the string supplied on the last USER command. If no USER commands has been issued, the string will be all blanks.
account	Account data. A standard_character string returned by FTP containing up to 80 characters. This is the string supplied on the last ACCT command. If no ACCT command has been issued, the string will be all blanks.
data_type	Current representation type. A standard_character string returned by FTP containing up to 80 characters. This is the string supplied on the last TYPE command, or the default if no command was issued.
file_structure	Current file structure. A standard_character string returned by FTP containing a single character. This is the character supplied on the last STRU command, or the default if no command was issued.
transfer_mode	Current transfer mode. A standard_character string returned by FTP containing a single character. This is the character supplied on the last MODE command, or the default if no command was issued.
pathname	Current pathname. A standard_character string returned by FTP containing up to 80 characters. This is the string supplied on the last RETR or STOR command.

reply_code — Reply to status request. Integer value is returned by FTP for all attempts regardless of success. Reply codes are defined in section 3.6.2.2.1; however, this reply code indicates only whether the given connection exists.

3.6.3.1.5 FTP_Send_Command (FTPSC) Routine (RFC-765)

The FTP_Send_Command routine shall send an FTP command over the specified FTP connection. FTP shall maintain user protocol state tables for each connection established. If a command that is invalid at a particular state is issued, FTP shall reject the command with an appropriate reply. Command parameters shall also be examined to ensure they conform to syntax specifications. No attempt is made to check syntax or state on commands not supported by the CYBER FTP. The calling format shall be:

CALL FTPSC (connection_id, command, no_wait, reply_code)

connection_id — FTP connection identifier. Integer value supplied by the application. This must be the identifier returned by FTP in the FTP_Connect request. The application uses this value to distinguish between different FTP connections simultaneously established.

command — FTP command to be issued. A standard_character string supplied by the application can contain any number of characters. The command string should include any required parameters. If the application does not supply a TELNET line termination sequence, FTP will append it to the end of the command.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call FTP_Accept_Reply to subsequently determine the reply.

reply_code — Reply from command. Integer value is returned by FTP for all attempts regardless of success. Reply codes are defined in section 3.6.2.2.1. If the application requires the message text, the FTP_Accept_Reply routine must be called. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

3.6.3.2 Control Statement (RFC-765)

The Control Statement interface shall be provided by FTP to terminal user jobs, batch jobs, and CYBER Control Language (CCL) procedures. If the control statement is invoked from a terminal, the program shall perform a dialogue with the user, displaying replies as they are received and sending commands as they are entered. All parameters are order-independent. The format of the control statement shall be:

```
FTP    (I=command_file, L=output_file, NA, PC=prefix_char)
```

command_file — Name of local file containing FTP commands. If omitted, file INPUT is assumed. If the file is assigned to a terminal, FTP may perform an interactive dialogue with the user, depending on the output_file. If I=0, FTP will take commands from subsequent lines of the current control statement file. FTP commands are distinguished from control statements by the prefix character specified by the PC parameter.

output_file — Name of file to receive FTP messages. If omitted, file OUTPUT is assumed. These messages consist of command replies from FTP processes with established connections. If the file is assigned to a terminal and command_file is also assigned to a terminal then FTP will perform an interactive dialogue with the user. If L=0, output messages are discarded.

NA — No-abort parameter. If specified, FTP will continue to read and process commands even if errors are encountered; normally FTP will abort the user if errors are encountered in a command file that is not assigned to a terminal. This parameter has no meaning for interactive users.

prefix_char — Command prefix character. This character must precede all commands which are embedded in the control statement file following the FTP statement. If omitted, asterisk (*) is assumed.

The complement of FTP standard commands available and their syntax is specified in section 3.6.2.1.1.

The commands and syntax described below apply only to control statement users. Users of the application-callable routines, and peer FTPs, must adhere to the CYBER FTP protocol standard and runtime calling conventions.

a. An OPEN command is defined. This command specifies the FTP hosts involved in the session. The format of the command is:

```
OPEN[SP][HOST][SP][HOST2]
```

At least one blank must separate each of the fields. If a host name itself must contain blanks, the entire host name must be delimited by

quotation marks (") or dollar signs ($) (the standard NOS delimiter). If the host name is delimited, and the host name itself contains the delimiter character, it must be entered twice in succession to distinguish the real use of the character from the delimiter.

The 'HOST' field is always required. It specifies the name of the remote host for the file transfer. If 'HOST2' is omitted, the file transfer will take place between the local host and the remote host. If 'HOST2' is supplied, the file transfer will take place between 'HOST' and 'HOST2'. A file transfer between two remote hosts is called a third-party transfer.

The internet address can be entered explicitly if the host name is unknown or if the site host name table does not list the host. An internet address must be enclosed in brackets (for example, [10.0.7.23]). Use of internet addresses is discouraged.

b. The OPEN command can only be entered as the first command, or after the CLOSE command. The CLOSE command terminates the FTP connections by sending a QUIT command to all remote hosts, without terminating the connection with the local host. CLOSE can be used to allow more than one FTP session during a single invocation of FTP. CLOSE takes no parameters.

c. Commands entered after the OPEN are directed to a specific FTP host that is either specified in the command or assumed from the default. The default host after an OPEN command is always the local host. The default can be changed using the HOST command. The format of the HOST command is

HOST[SP][HOST]

where 'HOST' is the name of the default host. It must match either the 'HOST' or 'HOST2' parameter specified in the last OPEN command, or must have the value 'LOCAL' or null to specify the local host. The 'HOST' parameter can contain a unique contiguous subset of the name specified in the OPEN command. For example, if the following open command were issued:

```
OPEN  SRI-NIC.ARPA  SRI-NCC.DDN
```

then 'HOST NIC' would refer to the first host, while 'HOST DDN' would refer to the second. The command 'HOST SRI' is in error because the string is non-unique.

d. A command can be directed to a non-default host by specifying '@host' before the command. For example:

```
OPEN  SRI-NIC.ARPA  SRI-NCC.DDN   [the default host is LOCAL]
@NIC  USER anonymous
@NIC  PASS
@NCC  USER anonymous
@NCC  PASS
HOST  NIC                        [the default host is SRI-NIC]
LIST                             [list catalog at host SRI-NIC]
```

A unique contiguous subset of characters can be entered for the host, just like the HOST command.

e. Certain commands always assume the local host regardless of the HOST command. These are QUIT, CLOSE, OPEN, HOST, and HELP. Of these commands, only the HELP command can ever be preceeded by the @host prefix; an @host preceeding any of the others is ignored.

f. The QUIT command operates the same as the CLOSE command, except that the local FTP connection is also terminated, the FTP control statement terminates, and control returns to NOS.

g. Commands and host names can be entered in upper and/or lower case. Care should be taken with commands directed to non-CYBER hosts, since upper and lower case may be significant for certain command parameters. Upper and lower case is insignificant for any FTP CYBER command or parameter. For example, the following two commands are equivalent:

```
OpEn  Sri-nic.arpa
Open  SRI-NIC.ArPa
```

h. Commands can be entered as the standard 4-character representation, the full representation, or as a unique abbreviation. A unique abbreviation is any truncated version (that is, trailing characters deleted) of a command that cannot be mistaken for another command. The valid commands are listed below:

Full	FTP Standard (4-character)	Minimum Unique Abbreviation
OPEN	n/a	O
CLOSE	n/a	C
HOST	n/a	HO
SEND	n/a	SE
RECEIVE	n/a	REC
RENAME	n/a	REN
HELP	HELP	HE or ?
QUIT	QUIT	Q
REINITIALIZE	REIN	REI
ABORT	ABOR	AB
SITE	SITE	SI
NOOP	NOOP	N
USER	USER	U
PASSWORD	PASS	PASS
ACCOUNT	ACCT	AC
PORT	PORT	PO
PASSIVE	PASV	PASV or PASSI

Full	FTP Standard (4-character)	Minimum Unique Abbreviation
MODE	MODE	M
TYPE	TYPE	T
STRUCTURE	STRU	STR
RESTART	REST	RES
STORE	STOR	STO
RETRIEVE	RETR	RET
LIST	LIST	L
APPEND	APPE	AP
RNFR	RNFR	RNF
RNTO	RNTO	RNT
DELETE	DELE	D

A command directed at a remote host that is not supported by CYBER FTP (for example, STAT) can be entered only as the four-character FTP standard - abbreviations are not supported for these commands.

i. A pair of commands has been defined to make it easier for the user to perform standard file transfers. They are SEND and RECEIVE. The format of the SEND command is:

```
SEND[SP][local_pathname][SP][remote_pathname]
```

This command has the same effect as would the two commands:

```
@local_host  RETR local_pathname
@remote_host STOR remote_pathname
```

The format of the RECEIVE command is:

```
RECEIVE[SP][local_pathname][SP][remote_pathname]
```

This command has the same effect as would the two commands:

```
@remote_host RETR remote_pathname
@local_host  STOR local_pathname
```

For both SEND and RECEIVE commands, at least one blank must separate each field. If a pathname contains a blank, the pathname must be delimited by quotations (") or dollar signs ($). If the pathname is delimited, and the pathname contains the delimiter character, each occurrence of the character must be repeated to distinguish from the delimiter character.

In third-party transfers, the local_host is the first host specified in the OPEN, and the remote-host is the second host specified in the OPEN.

An error is issued if the @host parameter is prefixed to SEND or RECEIVE.

Use of the STOR/RETR commands when SEND or RECEIVE will work is discouraged since the opportunity for user error is much greater with the former.

j. The RENAME command is defined as an alternative to the RNFR/RNTO sequence. Pathnames containing blanks must be delimited, just as for host names. The format of the command is:

```
RENAME[SP]new_pathname[SP]old_pathname
```

This command has the same effect as the following two commands:

```
RNFR  old_pathname
RNTO  new_pathname
```

Use of the RNFR/RNTO commands is discouraged since the opportunity for user error is much greater.

k. The prefix character is required before commands only if the control statement file is used (I=0 on FTP control statement). For interactive input, or other input files, it is not required but can be supplied.

l. Comments can be placed in the command stream by starting the line with two consecutive prefix characters. Extra blanks can be used between parameters, and blank lines between commands, to improve readability.

For example:

```
** Get DDN host name table.

      OPEN    SRI-NIC.ARPA

** Convert to standard NOS 6/12 display code.

      SITE    CS=ASCII

** Log into network host.

  @NIC USER    ANONYMOUS
  @NIC PASS    GUEST
       RECEIVE DDNHOST  <NETINFO>HOSTS.TXT
       QUIT
```

m. Commands can be continued by beginning continuation lines with two periods (..), but commands can only be broken between parameters; parameters themselves cannot be broken onto another line. For example:

```
    OPEN
..  SRI-NIC.ARPA
..  SRI-NCC.DDN
```

Note that in a command file, where the first character must be a prefix character, the above sequence would be as follows:

```
*  OPEN
*.. SRI-NIC.ARPA
*.. SRI-NCC.DDN
```

The excess blanks are not required. The above sequence could also be entered as:

```
*OPEN SRI-NIC.ARPA
*..SRI-NIC.DDN
```

n. If an interactive user break (for example, CTRL-T) is entered, the user is polled with the following:

```
FTP command XXXXXX in progress.  -or-  FTP Idle.
Do you wish to abort? (Y or N):
```

The first message issued depends on whether a command is being processed at the time the user entered the user break. If the user enters 'N' in response to the prompt, processing continues. If the user enters 'Y', an ABOR command is sent to both (or three) FTP hosts, followed by a QUIT. When these commands have completed, the FTP control statement aborts.

3.6.3.3 DDN/C170 Application Interface (RFC-765)

FTP shall utilize the DDN/C170 Application Interface to access DDN. Section 3.8.3 defines the DDN/C170 Application Interface primitives. The following services of the Application Interface shall be used:

1) TCP Primitives with optional TELNET protocol processing.
2) Host name server translation and error reporting.
3) DDN error logging.

3.6.3.4 NOS File System (RFC-765)

FTP shall use the NOS file system to access, create, and modify files as required by the FTP function. The NOS file functions are provided by the Permanent File Manager (PFM), Local File Manager (LFM), and Combined Input/Output (CIO) functions of NOS. The assembly language macro interfaces to these functions are defined in the NOS Reference Manual, Volume 4. The higher-level language interface to these macros, provided by SRVLIB, shall be used by FTP wherever possible.

3.6.3.5 FTP Peer Protocol (RFC-765)

FTP shall implement the communicate with remote FTP processes using the peer protocol specified in section 3.6.2.

3.6.3.6 Network Definition File (RFC-765)

FTP shall define its NAM application parameters via NDL configuration directives. The directives used are defined in subsection 3.6.8.

3.6.3.7 Operator Interface

FTP shall allow the operator to exercise some control over which messages are written to the log file. The following are the K-display HOP commands used to control messages written to the log file.

K.DE = FTP	Write error messages only to the log file
K.DB = FTP	Write debug and error messages to the log file
K.RS = FTP	Write statistics, debug, and error messages to the log file

Error messages are always written to the log file regardless of any operator intervention.

If the application has been loaded using NETIOD, the K.RS command will also restart AIP statistical gathering.

3.6.3.8 HELPLIB Interface

The Help text for FTP shall be maintained in HELPLIB, in the following CCL procedures:

FTP — Help for FTP control statement users. This procedure will define help for the FTP control statement parameters. This help is obtained by entering the HELPME,FTP command to NOS.

FTPUSER — Help for FTP local command users. This procedure will define help for FTP commands issued from a local FTP control statement. It will be organized from the viewpoint that the user is familiar with CYBER systems. This help can be accessed directly through the HELPME,FTPUSER command to NOS, or indirectly through the HELP command during a local interactive session with FTP.

FTPPEER — Help for FTP remote command users. This procedure will define help for FTP commands issued from a remote peer FTP. It will be organized from the viewpoint that the user is not familiar with CYBER systems. This help can be accessed directly through the HELPME,FTPPEER command to NOS, or indirectly through the HELP command during a session with a remote CYBER FTP (the help text originates from the remote CYBER).

Procedures within HELPLIB are maintained as a user library, beginning with a ULIB record and ending with an OPLD record, so the module that searches for a HELP topic can locate a procedure using the OPLD directory.

The help text within the procedures is divided into topics which are delimited using standard CCL syntax:

```
.HELP,topic1.
Help text for topic 1.

.HELP,topic2.
Help text for topic 2.

etc.
```

Each topic name is in upper case only, though the FTP/SMTP session user can enter the topic in both upper/lower case since FTP/SMTP will force it to upper case. A topic can be up to 10 characters in length. Topics can appear within the procedure in any order, since the structure of the procedure may require sets of help topics to appear in embedded sub-procedures (procedures following a .DATA directive). This will be necessary since CCL regards the .HELP topics as CCL parameters. The module that searches for a help topic must locate the appropriate procedure (for example, FTPPEER) and then do a sequential search for a '.HELP,topic.' line until an EOR is reached; no other assumptions can be made.

The help text itself is in upper/lower case 6/12 display code. Conversion to/from 8-bit ASCII for remote users is done by FTP/SMTP. No conversion is required for local users.

A help text line must be limited to 70 characters in length. When possible, a single topic should contain no more than 20 lines of text - if more lines are required, a sub-topic(s) should be created if possible. Since HELP is intended primarily to be helpful, the 20 line limit can be exceeded to any length, if necessary to maintain helpfulness. The limits are given to permit help text to fit on any terminal screen, and to spare hard-copy terminal users a long wait

during printouts. The last line in the help text body will always be a cross-reference entry 'See also: t1, t2, t3, t4, ..., tn.' where tn are other related topics. The help text body must always use correct English and correct punctuation.

3.6.4 Internal Interfaces (RFC-765)

The following subsections describe the interfaces between the major components of CYBER FTP. The internal interfaces identified are:

Protocol Interpreter to File Server
Application Interface to Protocol Interpreter
Control Statement to Application Interface

3.6.4.1 Protocol Interpreter to File Server Interface (RFC-765)

The Protocol Interpreter to File Server (PI/FS) interface consists of commands and associated responses passed via a NAM connection, where the commands are issued by the PI and responses are returned by FS. These commands instruct the FS about file transfer processing to be performed. Although many of the commands are similar or identical to the FTP protocol commands, PI does provide some processing for FS, as follows:

a) All commands/responses are issued in upper-case display code only. This frees the FS from character-set concerns for command processing. (RFC-765)

b) FTP Connection state tables are maintained by PI. The FS need be concerned only about the relevance of a command within its own state. (RFC-765)

c) All parameters have been broken out and processed. This frees the FS from syntax analysis concerns for command processing. In the case of NOS-specific information (for example, packname, user number), PI inserts this information into a standard PFM FET that is passed to FS. (RFC-765)

3.6.4.1.1 PI/FS NAM Connection (RFC-765)

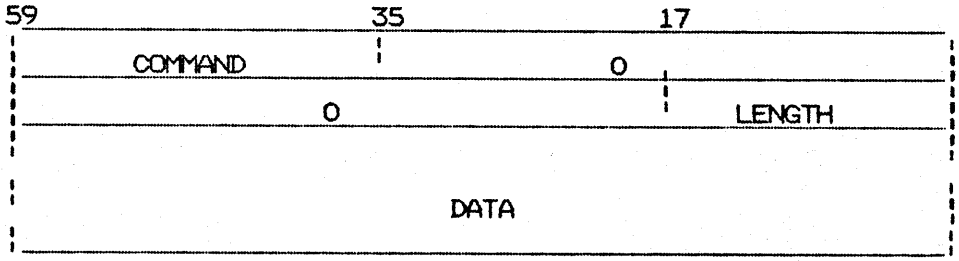
PI always initiates the NAM connection with FS. This connection request is issued as a result of an FTP user, either remote or local, establishing a connection with the local FTP host; if a local FTP user does not require a file transfer to or from the local host system (that is, a three-party transfer) then no file server is assigned. The standard connection acceptance/refusal criteria for NAM are employed. An FS will only accept a single connection, since an instance of FS is dedicated to a particular FTP user, whether local or remote. Instances of FS which have been initialized by NAM as part of start-up will NETON and wait for a connection request. File Servers will be started by the protocol interpreter. This is accomplished using the 'request startable' feature of NAM. When protocol interpreter requires a file server, it will request an A-A connection with FS; NAM in turn will start a copy of FS.

A single file server will usually be idle and rolled out waiting for a user so that users won't have to wait on file server job startup. When protocol interpreter is initiated, it will request a connection with the server. When the file server becomes assigned to a user, protocol interpreter will request another connection with a server, and that server will become the new idle server. If more than one user requests a server within a short period of time (1-2 seconds), then one of the users will have to wait until the another file server has initiated.

A given invocation of file server will only be used to handle a single user. That is, a file server is assigned to an FTP user and when that FTP user terminates (QUITs), the file server will also terminate. This happens as follows: a) when the user terminates, protocol interpreter waits for any in-progress file transfer to complete and then sends a quit message to FS, and b) File server closes its connection with FS when the quit command is received, and c) File server terminates (that is, 'goes away') when its connection with protocol interpreter is closed, whether by problems or deliberately as a result of the quit command. Note that the same file server remains assigned even if the user switches accounts with subsequent USER commands. Both the PI and the idle FS process are rolled out when not busy, to be rolled back in by NAM when connection requests are received.

3.6.4.1.2 PI/FS Commands (RFC-765)

Commands to FS are passed as data over the NAM connection. Only one command may be passed in a single QTPUT, and a single command may be spread over more than one QTPUT. The general format of a command is:



COMMAND: 4-character, left-justified, zero-filled, upper-case display code identifying the command.

LENGTH: 18-bit, right-justified, zero-filled, integer identifying the length of the command/data block. The length includes the command and length words.

DATA: Block of data associated with command. Format is specific to the command.

The following commands are defined:

Data Word	Data Description

The MODE command specifies the FTP file transfer mode as defined by FTP. The specified mode will remain in effect until the next MODE command or until the connection is closed.

1 of 1 Left-justified, zero-filled single character 'S', 'B', or 'C'. Character has same meaning as FTP MODE command.

The TYPE command specifies the FTP file transfer type as defined by FTP. The specified type will remain in effect until the next TYPE command or until the connection is closed.

1 of 2 Left-justified, zero-filled, two-character field consisting of 'AT', 'AC', 'AN', 'L', or 'I'. Characters have same meaning as FTP TYPE command.

2 of 2 Right-justified, 60-bit integer. Number passed with TYPE if TYPE = L, otherwise zero.

The STRU command specifies the FTP file transfer structure as defined by FTP. The specified structure will remain in effect until the next STRU command or until the connection is closed.

1 of 1 Left-justified, zero-filled character 'F' or 'R'. Character has same meaning as FTP STRU command.

The RETR command instructs FS to acquire the specfied permanent file and begin sending it across the TCP data connection according to the currently defined type, structure, and mode.

1 of n — Left-justified, zero-filled, two-character field defining the file type. The following values are allowed: 'IA' for indirect access file, 'DA' for direct access file, 'PF' for either direct or indirect access file, and 'LO' for local file. 'LO' is treated by FS the same as 'IA', since the PI creates a temporary indirect access file that is described in the FET below. If 'PF' is specified, FS will attempt to acquire the file either as direct or indirect access.

2 thru n — PFM formatted File Environment Table (FET). This table will contain all the parameters specified by the user for the file. In the case of a local file, it defines a temporary permanent file created by PI for the file transfer.

The STOR command instructs FS to create the specified permanent file and place into it data received from its TCP data connection according to the currently defined type, structure, and mode.

1 of n — File type. See RETR command. If 'PF' is specified, 'DA' will be assumed. If 'LO' is defined, STOR will create an indirect access file as defined by the FET below which PI will then use to create a local file. If a local file is specified, the destination must be the local user; remote users cannot create local files.

2 thru n — PFM FET. See RETR command.

The APPE command instructs FS to append data received from its TCP data connection to the specified permanent file, according to the currently defined type, structure, and mode.

1 of n — File type. See RETR command. If 'LO' is defined, APPE will append the data to an indirect access file created by PI and containing the local file data, which PI will then use to restore as the appended local file. If a local file is specified, the destination must be the local user; remote users cannot append local files.

2 thru n — PFM FET. See RETR command.

The RENM command instructs FS to rename the specified permanent file. This command is the combined FTP commands RNTO and RNFR.

1 of n — New permanent file name, left-justified, zero-filled.

2 thru n — PFM FET. See RETR command. This FET is the format required by the PFM CHANGE command.

The DELE command instructs FS to delete the specified permanent file.

1 of n — Unused. This word is maintained for consistency with the other PF commands. Local files cannot be deleted.

2 thru n — PFM FET. See RETR command.

The LIST command instructs FS to obtain a catalog of the specified user account and to send it over the data connection.

1 of n — Local Flag. Contains an integer. Value is 1 if the request is from a user on this host; zero otherwise.

2 thru n — PFM FET for CATLIST command.

The PORT command instructs FS to initiate a TCP data connection on the specified port. An existing connection will be terminated.

1 of 1 Internet address and port. A 48-bit, right-justified, zero-filled field containing the internet address and TCP port in 6 8-bit fields, as specified by IP protocol.

The PASV command instructs FS to initiate a passive TCP open on a TCP-assigned connection and to return that connection port as a response to the command.

1 of 1 Internet address and port. A 48-bit, right-justified, zero-filled field containing the internet address and TCP port in 6 8-bit fields, as specified by IP protocol. The port number will be zero; TCP will assign a port.

The USER command instructs FS to switch to the indicated user account. The charge and project numbers have not been validated.

1 of 3 User Number. A 1-7 character display code string, left-justified and zero-filled.

2 of 3 Family Name. A 1-7 character display code string, left-justified and zero-filled.

3 of 3 Password. A 1-7 character display code string, left-justified and zero-filled.

The ABOR command instructs FS to stop any ongoing transfer of data, close all network connections and terminate. No data is passed with the ABOR command.

The NAM upline supervisory message INTR/USR/R, sent from PI, will be interpreted as an ABOR command by FS.

The QUIT command instructs FS to accept no further commands from PI, finish any uncompleted commands, and terminate. No data is passed with the QUIT command.

The REIN command is functionally identical to the QUIT command.

The ACCT command instructs FS to charge file transfer activitiy to the specified account.

1 of 3 Charge Number. A 1-10 character display code string, left-justified and zero-filled.

2,3 of 3 Project Number. A 1-20 character display code string, left-justified and zero-filled.

The REST command instructs FS to perform the subsequent file transfer request as a restart from the location indicated in the data block.

1 of 3 File number. An integer value identifying the NOS logical file number to begin restart, counting from file 1.

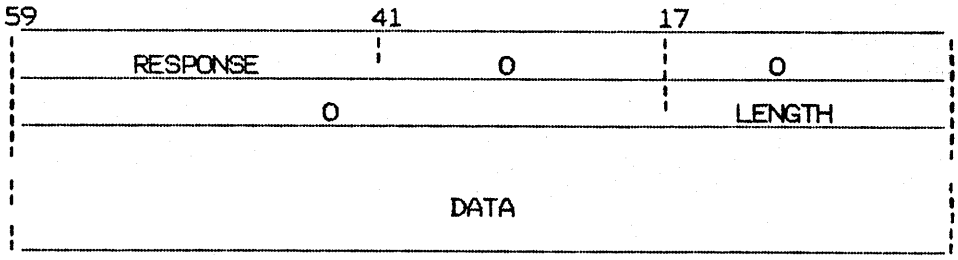
2 of 3 Record number. An integer value specifying the NOS logical record number within the file to begin restart, counting from record 1.

3 of 3 Byte number. An integer value specifying the byte number number within the record at which to begin restart. 8-bit bytes are assumed. Note that the total byte count cannot exceed 2**48-1 (281474976710655).

3.6.4.1.3 PI/FS Responses (RFC-765)

Responses to PI are passed as data over the NAM connection. Only one reponse may be passed in a single NETPUT, and a single response may be spread over more than one NETPUT. At least one response is required for each command sent by PI. If more than one response is required, these are sent as

intermediate replies prior to the final reply. The general format of a response is:



RESPONSE: 3-character, left-justified, upper-case display code identifying the FTP response number.

LENGTH: 18-bit, right-justified, zero-filled, integer identifying the length of the response/data block. The length includes the response and length words.

DATA: Block of data associated with response. Format is specific to the response.

The following responses are defined:

Data Word	Data Description
Response	110
1 of n	File number. Right justified, display code number of the file currently being received.
2 of n	Record number. Right justified, display code number of the file currently being received.
3 of n	Byte number -1. Right justified, display code portion of the byte number currently being received. This is the portion of the number greater than 10 decimal digits.

4 of n Byte number -2. Right justified, display code portion of the byte number currently being received. This is the least significant 10 decimal digits of the number.

5 of n Character count. 60-bit binary integer count of the number of characters in the following sender restart marker.

6 thru n Sender restart marker. Left justified string of characters received as restart marker.

Response 550

1 thru 3 PFM error text. The display code text returned by the permanent file manager explaining this error.

3.6.4.2 Application Interface (AI) to Protocol Interpreter (PI) (RFC-765)

The Application Interface to Protocol Interpreter (AI/PI) interface consists of requests and associated responses passed via a NAM connection, where the requests are issued by the AI and responses are returned by PI. AI issues requests containing FTP commands for processing by PI, or for forwarding to a remote FTP PI for processing. Minimal processing is performed by AI other than to package parameters supplied by an application into a request packet to be sent to PI, and to unpack responses received from PI to be returned to the application.

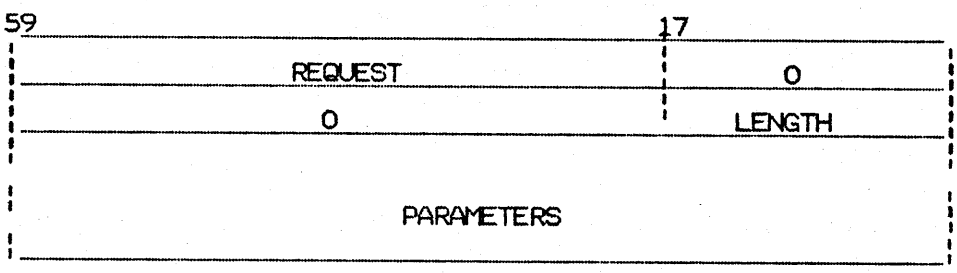
3.6.4.2.1 AI/PI NAM Connection (RFC-765)

AI always initiates the NAM connection with PI. This connection request is issued as a result of an FTP application issuing the first FTP_Connect request. Subsequent FTP_Connect requests are passed over the same AI/PI connection; that is, only a single NAM connection exists between AI and PI regardless of the number of FTP connections established by the application. The connection is terminated by PI when an FTP_Disconnect request is issued

for the last FTP connection established by the application. The Application Interface resides in the application field length, so an AI/PI connection will exist for each application in the system that has established FTP connections.

3.6.4.2.2 AI/PI Requests (RFC-765)

Requests to PI are passed as data over the NAM connection. Only one request may be passed in a single NETPUT, and a single request may be spread over more than one NETPUT. The general format of a request is:



REQUEST: 7-character, left-justified, zero-filled, upper-case display code identifying the request.

LENGTH: 18-bit, right-justified, zero-filled, integer identifying the length of the request/parameter block. The length includes the request and length words.

PARAMETERS: Block of parameters associated with request. Format is specific to the request. All input parameters are packed consecutively in the block with no more than one parameter per word. Unless otherwise specified, the parameter values are identical to those specified in subsection 3.6.3.1.

The following requests are defined:

Data Word	Data Description

The FTPC request solicits PI to establish an FTP connection with another FTP. If this is the first FTPC/FTPCS request issued since either the start of the program or since the termination of all previous FTP connections, then AI must establish the NAM connection with PI prior to issuing the request.

1-3 of 3	Host_address.

The FTPD request solicits PI to terminate a previously established FTP connection. After completion, AI also terminates the NAM connection with PI.

1 of 1	Connection_ID.

The FTPS request solicits PI to send the current status of an FTP connection.

1 of 1	Connection_ID.

The FTPSC request sends an FTP command to PI for processing. The command may be forwarded to a remote FTP or processed by the local PI depending on the command and the connection state. Multiple NETPUT requests may be required by AI to transfer the entire command to PI.

1 of n	Connection_ID.
2 thru n of n	Command.

In addition to requests issued for the standard AI routines, another intermediate request is defined which the user does not initiate. This request is the FTPLFS (FTP_Local_File_Saved request). If the application issues an FTP command (STOR or RETR) that involves a local file (that is, not permanent) then PI will issue a response to AI indicating that a local file

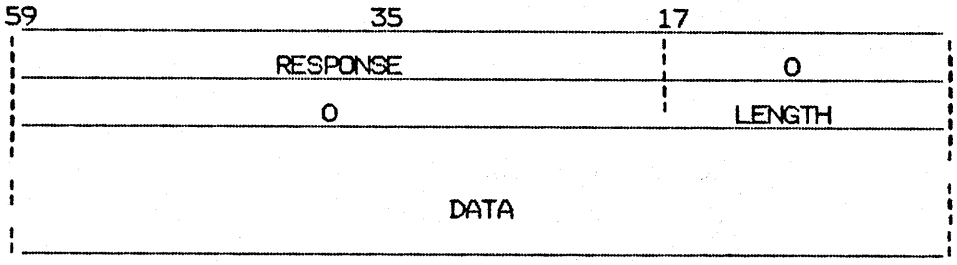
must be saved as a permanent file so that FS can access the file. AI creates the permanent file and then issues the FTPLFS request to PI to indicate completion.

1 of 9 — PFM Status code. This is the code returned by PFM in response to the permanent file request. If non-zero, the request was unsuccessful.

2 thru 9 of 9 — PFM error message. This is an error message returned by PFM if the permanent file request was unsuccessful.

3.6.4.2.3 AI/PI Responses (RFC-765)

Responses to AI are passed as data over the NAM connection. Only one reponse may be passed in a single NETPUT, and a single response may be spread over more than one NETPUT. One and only one response is returned for each request issued by AI. The general format of a response is:



RESPONSE: 7-character, left-justified, zero-filled, upper-case display code identifying the response.

LENGTH: 18-bit, right-justified, zero-filled, integer identifying the length of the response/data block. The length includes the response and length words.

DATA: Block of data associated with response. Format is specific to the response.

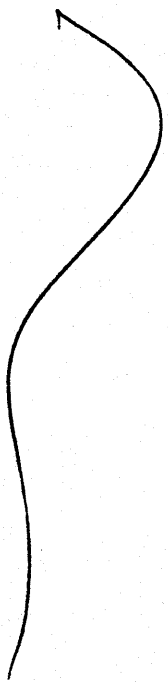
The following responses are defined:

Data Word	Data Description

The FTPC response returns the results of an FTPC request. If an FTPC request fails and no other FTP connections exist, PI will close the NAM connection after returning this response.

1 of 2	Connection_ID.
2 of 2	Reply_code.

The FTPD response returns the results of an FTPD request. If no other FTP connections exist, PI will close the NAM connection after returning this response.



Fill up text next printing.
Regenerate from here on.

1 of 9	Reply_code.
2 thru 9 of 9	Reply.

The FTPS response returns information requested in an FTPS request.

1 of 35	Reply_code.
2 of 35	In_progress.
3 thru 5 of 35	Host_address.
6 thru 14 of 35	User_name.
15 thru 23 of 35	Account.
24 of 35	Data_type.
25 of 35	File_structure.
26 of 35	Transfer_mode.
27 thru 35 of 35	Pathname.

The FTPSC response returns the results of the FTPSC request or FTPLFS request. If a command is issued for a local file operation, PI may return an FTPSCSF response instead.

1 of 1	Reply_code.

The FTPSCSF response is returned in reply to an FTPSC request if local file action is required. If this response is received, AI must perform a PFM REPLACE using the FET supplied in the response. AI must then issue the FTPLFS request with the results of the PFM REPLACE. PI will then issue and FTPSC response.

1 thru n	PFM FET to use in permanent file REPLACE request.

3.6.4.3 Control Statement to Application Interface (RFC-765)

The Control Statement to Application Interface is implemented by the AI routines. These routines are used by FTP C/S to issue FTP requests on behalf of an interactive terminal user or NOS batch job. This interface is defined in subsection 3.6.3.1.

3.6.5 Errors and Error Recovery (RFC-765, pg 262)

There is no requirement for detecting bits lost or scrambled in data transfer since this level of error processing is provided by TCP. However, the FTP shall provide a restart capability as described below.

3.6.5.1 FTP Restart (RFC-765, pg 262)

The FTP shall provide a restart capability for file transfers in the BLOCK and COMPRESSED modes only. To support this capability, the FTP shall: (RFC-765, pg 262)

a. Insert a special marker code in the data stream that will have meaning to a receiver that implements restart. (RFC-765, pg 262)

b. Use printable characters for the marker code in the default or negotiated language of the TELNET connection (ASCII). (RFC-765, pg 262)

c. Recognize marker codes received from a sender process and issue restart commands to the sender to restart the process at the marker point. (RFC-765, pg 263)

d. Restart a sending process upon receipt of a restart command from a receiver process. (RFC-765, pg 263)

3.6.6 Data Structures (RFC-765, pg 250)

3.6.6.1 Data Representation and Storage (RFC-765, pg 250)

The following data type representations types shall be provided by FTP.

a. ASCII - This is the default type which must be supported by FTP implementations. This data type shall also allow the user to specify: (RFC-765, pg 251)

   1. ASCII NON-PRINT - default format with no vertical format information used primarily for files destined for storage or processing. (RFC-765, pg 252)

   2. ASCII TELNET FORMAT CONTROLS - specifies that the file contains the TELNET vertical format controls of [CR], [LF], [NL], [VT], or [FF]. (RFC-765, pg 252)

   3. ASCII CARRIAGE CONTROL - specifes that the file contains the following vertical control characters: (RFC-765, pg 252)

      blank - move paper up one line
      0 - move paper up two lines
      1 - move paper to top of next page
      + - no movement; that is, overprint

b. IMAGE - Data in this format are sent as contiguous bits. It is intended that this format be used for efficient storage and retrieval of files and for transfer of binary data. (RFC-765, pg 253)

FTP shall assume the default CYBER ASCII representation is in 6/12-bit display code. The SITE command shall be used to specify 8-bit ASCII or 6-bit display code.

3.6.6.2 FTP File Structures (RFC-765, pg 254)

FTP shall support the following file structures:

a. File-Structure - files where the file is considered to be a continuous sequence of data bytes. (RFC-765, pg 254)

b. Record-Structure - files where the file is made up of sequential records. (RFC-765, pg 254)

3.6.6.3 FTP Numerical List of Responses (RFC-765, pg 280)

A numerical list of FTP responses can be found in RFC-765, page 280.

3.6.6.4 FTP Command Syntax (RFC-765, pg 285)

The FTP command syntax can be found in RFC-765, pg 285.

3.6.7 Equipment Configuration (RFC-765)

FTP has no special equipment configuration requirements.

3.6.8 Installation (RFC-765)

The following subsections define the installation procedures and parameters for FTP.

3.6.8.1 FTP Installation Procedure (RFC-765)

FTP is installed using the FTP build procedure and build verification is performed using the VFTP procedure. FTP should be treated as a Group 3 job as defined in the NOS Installation Handbook. It depends upon the DDN Supervisor Installation defined in subsection 3.8.8. The FTP product PL is maintained in MODIFY format. The following files are created or updated by the build procedure: JOBFTIP, JOBFTPS, HELPLIB(updated), DNILIB(updated).

3.6.8.2 FTP Installation Parameters (RFC-765)

The following installation parameters can be tailored for specific sites. They are located in common deck COMVIPR.

IPDLCS — Default Local Character Set. This is the default character set for commands, responses and files at the local CYBER host. Possible values are ASCII, ASCII63, and ASCII64 as defined by the FCOPY statement. The default value is ASCII.

IPFBTO — File Busy Timeout. This is the maximum time that FTP will wait for a busy file when the NA or WB parameter is specified. After this period, the file transfer attempt will terminate. The default is 600.

IPFCTO — FTP Command Timeout. This is the maximum time in seconds that FTP will wait for a remote FTP to respond to a command. After this period, the connection will be aborted. The default is 600.

IPMBUF — Maximum Buffer Size. This is the buffer size used for file transfers. The minimum size is the NAM downline block size (255 words default). The default value is 1024B.

IPRTRY — Retry Limit. This is the maximum retry limit for errors before FTP terminates the connection.

The following modify symbols can be specified using the *DEFINE directive to influence the FTP build process:

DEBUG If defined, FTP will generate AIP trace messages.

IMS If defined, the FTP listing will contain Internal Maintainance information.

STATS If defined, FTP will generate AIP statistics messages.

3.6.8.3 Network Startup Master File (RFC-765)

The following directives must be placed in the Network Startup Master File if automatic initiation of FTP is required.

```
JOB(JOBFTPI,FTPI)
```

3.6.8.4 Network Definition File Directives (RFC-765)

The following table lists the DDN applications and their configuration parameters:

Name	Description	Number of Copies (MXCOPYS)	Request Startable	Privileged
FTPI	FTP Protocol Interpreter	1	yes	yes
FTPS	FTP File Server	any(5)	yes	yes
FTP	FTP Control Statement	any(5)	no	yes
FTPA	FTP User Application	any(1)	no	no

The values for MXCOPY can be adjusted by the site to control FTP usage. A small number can mean that FTP users will have to wait for resources while a large number can tie up TCP and NAM connections and machine resources, affecting other users of the system.

Additionally, FTP application users (FTPA) must provide OUTCALL directives to FTP for their applications.

3.6.8.5 FTP Transmittal

No other DDN CPCI transmittal can depend upon this transmittal.

a. FTP Program Library (FTPnnnn). This is an UPDATE formatted program library containing the FTP application interface routines, Protocol Interpreter, File Server and FTP Control Statement. The format of this program library is described later. The following object is generated from this PL:

Name	Type	Location	Description
FTPI	Program	System	FTP Protocol Interpreter
FTPS	Program	System	FTP File Server
FTP	Program	System	FTP Control Statement
DNILIB	Library	System	FTP Additions to DDN Interface Library
NAMSTRT	Procedures	un=NETOPS	NAM Startup Proc. Update for FTPS/FTPI (JOBFTPI,JOBFTPS)
HELPLIB	CCL Library	un=LIBRARY	FTP Help Procedures Update to HELPLIB (FTP/FTPPEER/FTPUSER)

b. FTP Installation procedure (FTP). This procedure follows DECKOPL conventions to install FTP.

c. FTP Installation Verification procedure (VFTP). This procedure is to be executed on a newly-built system to verify that FTP has been properly installed. It does not verify that FTP actually works. It follows DECKOPL verification conventions.

3.6.8.6 PL Structure

a. Where deck lists are described as 'organized', it means that the decks are arranged in a SIMPLE manner logical with the structure of the program. For example, the main program followed by all primary routines in alphabetical order followed by all minor routines in alphabetical order. If no structure can be identified, 'organized' degenerates to 'alphabetized', never to 'random' or 'complex'.

b. FTP PL Structure.

```
INFO
HISTORY
[alphabetized common decks]
-LIBSYMP-  [contains *CWEOR]
[SYMPL library routines, alphabetized]
-LIBCOMP-  [contains *CWEOR]

[COMPASS library routines, alphabetized]
-SYMPL-    [contains *CWEOR]
-PISYMP-   [empty deck]
[FTPI SYMPL routines, organized]
-FSSYMP-   [empty deck]
[FTPS SYMPL routines, organized]
-CSSYMP-   [empty deck]
[FTP CS SYMPL routines, organized]
-COMPASS-  [contains *CWEOR]
-PICOMP-   [empty deck]
```

```
[FTPI COMPASS routines, alphabetized]
-FSCOMP-    [empty deck]
[FTPS COMPASS routines, alphabetized]
-CSCOMP-    [empty deck]
[FTP CS COMPASS routines, alphabetized]
-NAMSTRT-   [contains *CWEOR]
JOBFTPI
JOBFTPS
-HELPLIB-   [contains *CWEOR]
FTP
-HEOR1-     [contains *CWEOR]
FTPPEER
-HEOR2-     [contains *CWEOR]
FTPUSER
-ENDPL-     [empty deck]
```

NOTE: All non-library SYMPL/COMPASS decks are compiled into a large library. The FTPI, FTPS, and FTP main programs are separately loaded using a LIBLOAD from that library.

c. Inter-Component Common Decks

DDN System-Wide (defined in DDNTEXT)

COMVDDN	DDN General Runtime Interface Definitions
COMVDIP	DDN System Installation Parameters
COMVFET	CIO FET Definitions
COMVFIT	CRM FIT Definitions
COMVIPC	IP Interface Definitions
COMVNCT	DDN NCT Definitions
COMVSYS	CYBER System Definitions
COMVTCP	TCP Interface Definitions
COMVTEL	TELNET Interface Definitions

COMVAPC	FTP Application to Protocol Interpreter Interface
COMVFTP	FTP Definitions
COMVIPF	FTP Installation Parameters
COMVPFC	FTP Protocol Interpreter to File Server Interface
COMVSSJ	FTP NOS Validation Interface

3.7 Simple Mail Transfer Protocol (SMTP) (RFC -821/822)

3.7.1 SMTP Abstract (RFC-821)

The purpose of SMTP is to transfer mail reliably and efficiently, directly from the sending user's host to the receiving user's host when the two hosts are connected to the same transport service.

The SMTP will be implemented on the CYBER host to provide sender and recipient SMTP functions for DDN users. CYBER SMTP, in conjunction with remote host SMTP facilities, will provide the CYBER user and remote host user with the capability to send and receive mail messages via DDN. Actual delivery of the mail to a local mailbox is provided by a mail server application that uses the SMTP services. The model for SMTP usage is depicted in Figure 3.7-1.

SMTP is composed of four major components. The SMTP components are:

1) SMTP Protocol Interpreter (PI or SMPI)
2) SMTP Application Interface (AI)
3) SMTP Control Statement (CS or SMPC)
4) SMTP Mail Server (SMPS)

Protocol Interpreter is the primary SMTP module. It implements the protocol and performs all communications with remote SMTP peers. In addition, it provides mail services to the Control Statement and Mail Server through the

Application Interface. PI runs as a privileged system application program and communicates with other SMTP components via QTRM/NAM and permanent files. Communication with SMTP peers is performed using the C170 DoD Interface component of the DDN Gateway CPCI. PI establishes TCP connections with remote peers to send outbound mail to remote hosts, and accepts TCP connections from remote peers to receive inbound mail from remote hosts.

Application Interface provides a common interface between user or SMTP application programs and the PI. The SMTP Control Statement and Mail Server components use AI, as do site-specific mail servers or user applications that wish to send mail. AI is a set of library procedures on DNILIB that are accessed using direct procedure calls.

Control Statement provides a rudimentary method for interactive or batch users to send mail to other SMTP hosts, or to SMTP users on the local host. Since NOS does not provide a standard mail interface, CS is provided to perform this function. CS only sends mail and does not have provisions for reading incoming mail. Specific sites may choose to integrate their local mail systems with SMTP using AI, in which case CS is not required.

Mail Server provides a rudimentary method for the delivery of mail destined for users on the local host. Since NOS does not provide a standard mail interface, SMPS is provided to perform this function. SMPS places inbound mail into a user mail file which can later be read by the user. An installation option is available that makes the mail file compatible with the existing SES mail system.

The various SMTP components communicate via NAM connections that are depicted in Figure 3.7-2. Each of the interfaces are numbered and correspond to the summaries below:

1) Application programs issue requests to, and receive information from, PI through QTRM connections established by AI. Two SMTP components, the Control Statement and Mail Server, are examples of such applications but user applications may also use mail services. Several instances of each kind of program, i.e. Control Statement, may be active at one time so that several connections (1a, 1b, 1c, 1d) of this type may exist at one time.

2) PI registers with the DoD AI to get host name translations, obtain common host information, and log messages. This registration is obtained through the DoD AI, which creates a QTRM/NAM connection with the DDN Supervisor. Only one such connection is created.

3) PI opens active and passive TCP connections with remote SMTP peers, using the DoD AI. DoD AI in turn opens QTRM/NAM connections with the NP DoD G/W in the CDCNET MDI to obtain access to TCP services residing there. Many SMTP peer connections can exist at one time.

4) Mail data is exchanged between SMTP components, and stored for later retrieval, using permanent files maintained by NOS and CRM. Access to these files is handled by SMTP AI for PI. There is an instance of SMTP AI for each instance of an SMTP User Application, SMTP Mail Server, or SMTP Control Statement.

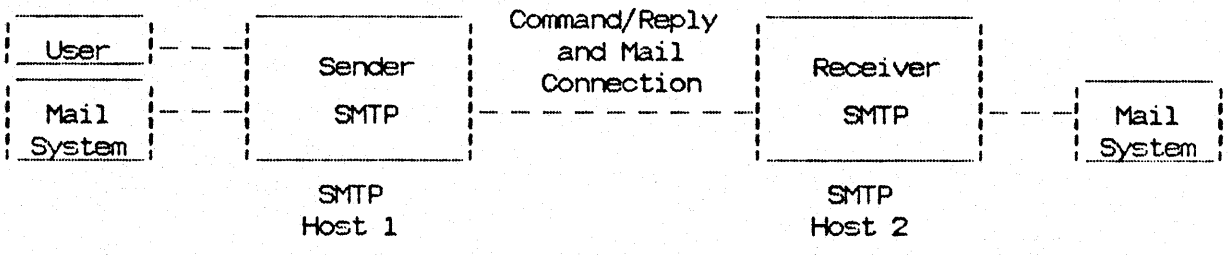


Figure 3.7-1. Model for SMTP Use

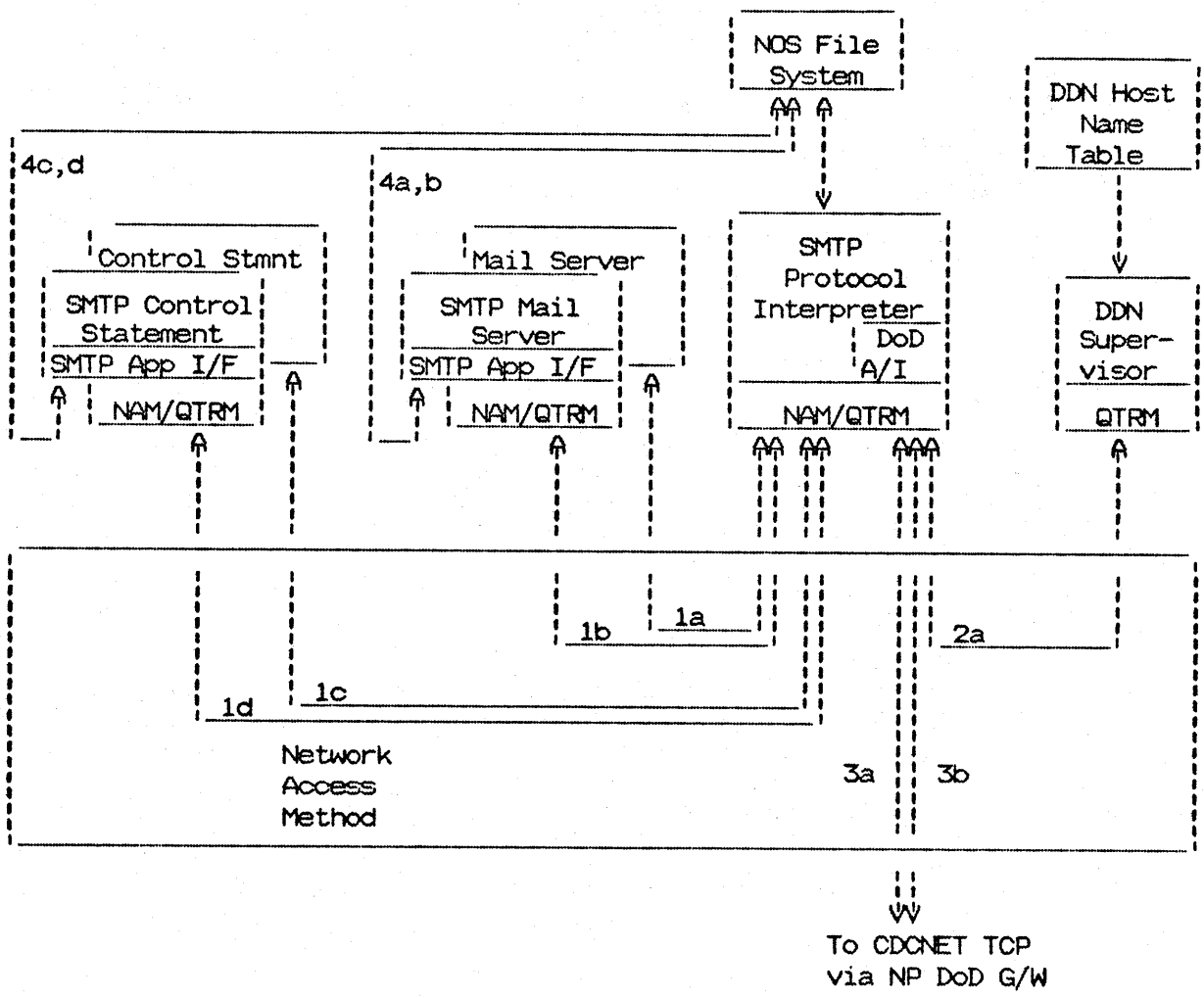


Figure 3.7-2. SMTP Functional Relationships

3.7.2 SMTP Description (RFC-821, pg 2)

Figure 3.7-1 illustrates a model of the SMTP in which the sender of mail establishes a two-way transmission channel to a receiver of mail. SMTP commands are generated by the sender and sent to the receiver and SMTP replies are sent from the receiver to the sender in response to the commands.

Once the transmission channel has been established and the sender and receiver have agreed that mail can be sent and delivered to the recipient, the sender sends the mail data. The sender then indicates the end of mail transfer and the receiver responds with a reply that the mail has been received and processed.

The following subsections specify the requirements for the protocol between a sender and receiver SMTP.

3.7.2.1 SMTP Functions (RFC-821, pg 4)

The CYBER SMTP shall provide sender and receiver electronic mail service such that SMTP mail transactions may be accomplished.

3.7.2.1.1 SMTP Sender Function (RFC-821, pg 4)

As a sender, the CYBER SMTP shall:

a. Initiate a new mail transaction by sending a MAIL command to the receiver that identifies the sender. (RFC-821, pg 4)

b. Send a RCPT command(s) to the receiver to identify the mail recipient(s). (RFC-821, pg 4)

c. Send a DATA command to the receiver followed by the mail data and end of mail indicator. (RFC-821, pg 5)

d. Recognize and process receiver responses to a, b, and c above. (RFC-821, pg 4,5)

3.7.2.1.2 SMTP Receiver Function (RFC-821, pg 4)

As a receiver, the CYBER SMTP shall:

a. Recognize the MAIL command, reset state tables and buffers, and send a 250 OK reply to the sender if accepted or an error reply if not accepted. (RFC-821, pg 4)

b. Recognize the RCPT command(s) and send a 250 OK reply if accepted or an error reply if not accepted. (RFC-821, pg 4)

c. Recognize the DATA command and send a 354 intermediate reply if accepted and consider all succeeding lines to be message text. When the end of mail indicator is received and stored a 250 OK reply shall be sent to the sender. (RFC-821, pg 4)

d. If the DATA command is not accepted an error reply shall be sent. (RFC-821, pg 5)

e. Process the recipient(s) mail data. (RFC-821, pg 5)

   1. The receiver SMTP shall append mail data to the mail buffer. (RFC-821, pg 20)

   2. The receiver SMTP shall accept all of the 128 ASCII characters as mail data. (RFC-821, pg 20)

   3. The receiver SMTP shall recognize the end of mail data as the ASCII character sequence of [CRLF]"."[CRLF]. (RFC-821, pg 20)

   4. The receiver SMTP shall ensure that information in the reverse-path, forward-path, and mail data is cleared after delivery of the mail data. (RFC-821, pg 20)

5. The receiver SMTP shall insert at the beginning of the mail data a time stamp line (see example in 7 below) for all accepted messages delivered to a mailbox or relayed to the next SMTP. (RFC-821, pg 20)

6. The receiver SMTP shall include the following information in the time stamp line (see example in 8 below): (RFC-821, pg 20)

   Identity of host sending the mail.

   Identity of host receiving the mail and inserting the time stamp.

   Date and time the mail was received,

7. The receiver SMTP that makes the final mail delivery shall insert at the beginning of the mail data a return path line that preserves the information in the reverse-path from the MAIL command. (RFC-821, pg 20,21)

   NOTE: The following example of return path and received time stamps is provided for information.

```
Return-path: <@GHI.ARPA,@DEF.ARPA,@ABC.ARPA:JOE@ABC.ARPA>
Received: from GHI.ARPA by JKL.ARPA ; 27 Oct 81 15:27:39 PST
Received: from DEF.ARPA by GHI.ARPA ; 27 Oct 81 15:15:13 PST
Received: from ABC.ARPA by DEF.ARPA ; 27 Oct 81 15:01:59 PST
Date: 27 Oct 81 15:01:01 PST
From: Joe@ABC.ARPA
Subject: Improved Mailing System Installed
To: SAM@JKL.ARPA

This is to inform you that ...
```

8. The receiver SMTP shall provide a response for partially successful processing following the end of mail data. The response shall be as follows: (RFC-821, pg 22)

   Send an OK reply in response to the DATA COMMAND. (RFC-821, pg 22)

   Send an undeliverable mail notification to the originator of the mail that lists the recipients for which mail could not be delivered using the MAIL COMMAND. (RFC-821, pg 22)

3.7.2.1.3 SMTP Open/Close Function (RFC-821, pg 13)

a. The sender SMTP shall initiate the transmission channel connection, wait for a 220 reply from the receiver, and send a HELO command to the receiver to identify itself. (RFC-821, pg 13)

b. The receiver SMTP shall send a 220 reply to the sender when the transmission channel connection is established and shall send 250 reply to the sender in response to the HELO command to identify itself. (RFC-821, pg 13)

c. The sender SMTP shall send a QUIT command to the receiver SMTP to close the connection. (RFC-821, pg 13)

d. The receiver SMTP shall send a 221 reply to the sender upon receipt of a QUIT command and then close the connection. (RFC-821, pg 13)

3.7.2.1.4 This subsection is blank to maintain paragraph numbering.

3.7.2.1.5 SMTP Command Syntax (RFC-821, pg 19)

The SMTP commands define the mail transfer or the mail system function requested by the user. The syntax of all SMTP commands shall be as specified in a. through g. below.

a. SMTP commands shall begin with a command code that shall consist of four alphabetic characters followed by an ASCII space. (RFC-821, pg 19,27)

b. Upper and lower case alphabetic characters shall be recognized equally in the command code. (RFC-821, pg 27)

c. An argument field, if present, shall consist of a variable length character string following the command code. (RFC-821, pg 28)

d. Upper and lower case alphabetic characters shall be recognized equally in an argument field. (RFC-821, pg 27)

e. SMTP commands without an argument field shall be terminated with an ASCII space followed by the ASCII character sequence of Carriage Return and Line Feed [CRLF]. (RFC-821, pg 19,27)

f. SMTP command with an argument field shall be terminated with the ASCII character sequence of [CRLF] immediately following the argument field. (RFC-821, pg 28)

g. No action shall be taken by the receiver of the SMTP command until the [CRLF] has been received. (RFC-821, pg 28)

NOTE: The actual command codes, argument fields, and syntax of all SMTP commands to be implemented are stated in subsection 3.7.2.1.5.1.

3.7.2.1.5.1 SMTP Commands (RFC-821, pg 19)

The SMTP commands described in subsections a. through h. below shall be implemented. (RFC-821, pg 29)

NOTE: In the COMMAND FORMAT subsections a.1 through h.1, items enclosed in brackets [] have the following meaning:

[SP] - represents one ASCII space.

[...] - represents a command argument.

[CRLF] - represents the ASCII character sequence of Carriage Return Line Feed.

a. HELLO COMMAND (RFC-821, pg 19)

This command shall be sent by the SMTP sender to the SMTP receiver to identify the sender to the receiver. (RFC-821, pg 19)

1. COMMAND FORMAT - HELO[SP][DOMAIN][CRLF] (RFC-821, pg 29)

HELO - The command code shall be the character string "HELO" in upper and/or lower case ASCII characters. (RFC-821, pg 19,27)

DOMAIN - The DOMAIN argument shall be the name of the sender host computer. (RFC-821, pg 64)

COMMAND EXAMPLE - HELO CDC-NOS.SNVL.ARPA

2. The sending of this command and receipt of an OK reply from the receiver shall confirm that the sender and receiver are in the initial state in which there is no transaction in progress and all state tables and buffers are cleared. (RFC-821, pg 19)

b. MAIL COMMAND (RFC-821, pg 20)

This command shall be sent by the SMTP sender to the SMTP receiver to initiate a mail transaction in which the mail data will be delivered to one or more recipients. (RFC-821, pg 20)

1. COMMAND FORMAT - MAIL[SP] "FROM:"[reverse-path][CRLF] (RFC-821, pg 29)

   MAIL - The command code shall be the character string "MAIL"in upper and/or lower case ASCII characters. (RFC-821, pg 20,27)

   FROM:[reverse-path] - The argument shall consist of the ASCII characters FROM: followed immediately by the reverse-path. (RFC-821, pg 20)

   reverse-path - The reverse-path shall consist of an optional list of hosts and a required sender mailbox. The sender mailbox format is defined by the local mail server and shall not be restricted by SMTP, beyond the restrictions imposed on reverse-path. (RFC-821, pg 20)

   The list of hosts is a reverse source route and indicates that the mail was relayed through each host on the list with the first host in the list as the most recent relay. This list is used as a source route to return non-delivery notices to the sender. Each relay host adds its host name to the beginning of the list and host names are separated with commas. The last host name in the list is separated from the sender mailbox with a colon ":".

   The mailbox is the sender's mailbox and can be thought of as the return address of the originator of the mail.

COMMAND EXAMPLE - MAIL FROM: <Yuma@CDC-NOS.SNVL.ARPA>

- MAIL FROM: <@HOST1.ARPA,@HOST2.ARPA:Yuma@CDC-NOS.SNVL.ARPA>

c. RECIPIENT COMMAND (RFC-821, pg 20)

This command shall be sent by the sender SMTP to the receiver SMTP to identify an individual recipient of mail data with multiple recipients specified by multiple use of this command. (RFC-821, pg 20)

1. COMMAND FORMAT - RCPT[SP]"TO:"[forward-path][CRLF] (RFC-821, pg 29)

   RCPT - The command code shall be the character string "RCPT" in upper and/or lower case ASCII characters. (RFC-821, pg 20,27)

   TO:[forward-path] - The argument shall consist of the ASCII characters TO: followed immediately by the forward-path. (RFC-821, pg 20)

   forward-path - The forward-path shall consist of an optional list of hosts and a required destination mailbox. (RFC-821, pg 20)

   When the list of hosts is present, it is a source route and indicates that mail must be relayed to the next host on the list. When mail is relayed, the relay host must remove itself from the beginning forward-path and put itself at the beginning of the reverse-path. When mail reaches its ultimate destination, the forward-path contains only a destination mailbox and the receiver SMTP inserts the mail into the destination mailbox. See 3.7.2.1.4 for an example.

COMMAND EXAMPLE - RCPT TO: <Yuma@CDC-NOS.SNVL>

- RCPT TO: <@HOST1.DDN,@HOST2.DDN:Yuma@CDC-NOS.SNVL>

d. DATA COMMAND (RFC-821, pg 21)

This command shall be sent by the sender SMTP and shall cause the receiver SMTP to interpret all following lines up to the end of mail indicator, [CRLF].[CRLF], as data from the sender SMTP.

1. COMMAND FORMAT - DATA[SP][CRLF] (RFC-821, pg 29)

   DATA - The command code shall be the character string "DATA" in upper and/or lower case ASCII characters. (RFC-821, pg 20,27)

   EXAMPLE - DATA

e. RESET COMMAND (RFC-821, pg 25)

This command shall be sent by the sender SMTP and shall cause the current mail transaction to be aborted.

1. COMMAND FORMAT - RSET[SP][CRLF] (RFC-821, pg 29)

   RSET - The command code shall be the character string "RSET" in upper and/or lower case ASCII characters. (RFC-821, pg 25,27)

   EXAMPLE - RSET

2. The receiver SMTP shall send an OK reply upon receipt of this command. (RFC-821, pg 25)

3. The receiver SMTP shall discard all received mail data and shall clear all buffers and state tables upon receipt of this command. (RFC-821, pg 25)

4. The sender shall discard all mail data and shall clear all buffers and state tables upon receipt of this command. (RFC-821, pg 25)

f. HELP COMMAND (RFC-821, pg 25)

This command shall cause the receiver SMTP to send helpful information regarding SMTP implementation to the sender of the HELP command.

1. COMMAND FORMAT - HELP[SP][KEY WORD][CRLF] (RFC-821, p29)

HELP - the command code shall be the character string "HELP" in upper and/or lower case ASCII characters. (RFC-821, pp25,27)

KEY WORD - The KEY WORD argument shall be an ASCII upper and/or lower case character string that indicates the type of help wanted.

The text for HELP topics are taken from CCL procedures. Three different HELP procedures are supported:

SMTP Help for SMTP control statement users. This procedure will define help for the SMTP control statement parameters. This help is obtained by entering the HELPME,SMTP command to NOS.

SMPUSER Help for SMTP local command users. This procedure will define help for SMTP commands issued from a local SMTP control statement. It will be organized from the viewpoint that the user is familiar with CYBER systems. This help can be accessed directly through the HELPME,SMPUSER command to NOS, or indirectly through the HELP command during a local interactive session with SMTP.

SMPPEER Help for SMTP remote command users. This procedure will define help for SMTP commands issued from a remote peer SMTP. It will be organized from the viewpoint that the user is not familiar with CYBER systems. This help can be accessed directly through the HELPME,SMTPPEER command to NOS, or indirectly through the HELP command during a session with a remote CYBER SMTP (the help text originates from the remote CYBER). Note that a local CYBER SMTP does not permit an interactive session with a remote SMTP, but other systems may, and the user is not prohibited from logging directly into the SMTP from a DDN terminal (that is, on a TAC).

Procedures within HELPLIB are maintained as a user library, beginning with a ULIB record and ending with an OPLD record, so the module that searches for a HELP topic can locate a procedure using the OPLD directory.

The help text within the procedures is divided into topics which are delimited using standard CCL syntax:

```
.HELP,topic1.
Help text for topic 1.
```

```
.HELP,topic2.
Help text for topic 2.

etc.
```

Each topic name is in upper case only, though the SMTP session user can enter the topic in both upper/lower case since SMTP will force it to upper case. A topic can be up to 10 characters in length. Topics can appear within the procedure in any order, since the structure of the procedure may require sets of help topics to appear in embedded sub-procedures (procedures following a .DATA directive). This will be necessary since CCL regards the .HELP topics as CCL parameters. The module that searches for a help topic must locate the appropriate procedure (for example, SMPPEER) and then do a sequential search for a '.HELP,topic.' line until an EOR is reached; no other assumptions can be made.

The help text itself is in upper/lower case 6/12 display code. Conversion to/from 8-bit ASCII for remote users is done by FTP/SMTP. No conversion is required for local users.

A help text line must be limited to 70 characters in length. When possible, a single topic should contain no more than 20 lines of text - if more lines are required, a sub-topic(s) should be created if possible. Since HELP is intended primarily to be helpful, the 20 line limit can be exceeded to any length, if necessary to maintain helpfulness. The limits are given to permit help text to fit on any terminal screen, and to spare hard-copy terminal users a long wait during printouts. The last line in the help text body will always be a cross-reference entry 'See also: t1, t2, t3, t4, ..., tn.' where tn are other related topics. The help text body must always use correct English and correct punctuation.

The topics to be supplied include a general help discussion (.HELP. command with no topic), a topic called TOPICS which list all possible help topics, PARAMETERS which lists all possible parameters, REMOTEUSER which provides remote user-specific info, LOCALUSER which provides local user-specific info, ERRORS which provides general error help, a set of Exxxx topics for specific errors where xxxx is a specific error code (the text will discuss possible corrections, not just reiterate the original error message text), EXAMPLE which gives an example of a file/mail transfer, and MINIMAL which describes the minimal set of commands and parameters to do a rudimentary mail transfer. There will also be at least one topic for each command and each parameter, plus topics on general subjects such as record structures, character sets, examples, etc.

g. NOOP COMMAND (RFC-821, pg 26)

This command shall not affect any parameters or previously entered commands and the only action to be taken is that the receiver of this command shall send an OK reply.

1. COMMAND FORMAT - NOOP[SP][CRLF] (RFC-821, pg 29)

NOOP - The command code shall be the character string "NOOP" in upper and/or lower case ASCII characters. (RFC-821, pg 26,27)

EXAMPLE - NOOP

h. QUIT COMMAND (RFC-821, pg 26)

This command shall be sent by the sender SMTP and shall cause the reciver SMTP to send an OK reply and then close the transmission channel.

1. COMMAND FORMAT - QUIT[SP][CRLF] (RFC-821, pg 29)

   QUIT - The command code shall be the character string "QUIT" in upper and/or lower case ASCII characters. (RFC-821, pg 26,27)

   EXAMPLE - RSET

3.7.2.2 SMTP Replies (RFC-821, pg 34)

Replies to SMTP commands shall be used to ensure the synchronization of requests and actions in the process of mail transfer, and to guarantee that the sender-SMTP always knows the state of the receiver-SMTP.

NOTE: In the following paragraphs [SP] means an ASCII space and [CRLF] means an ASCII Carriage Return Line feed.

a. At least one reply shall be generated by the receiver SMTP for each command received. (RFC-821, pg 34)

b. A single line SMTP reply shall consist of a 3 digit code transmitted as three alphanumeric characters followed by a [SP], followed by one line of text, followed by a [CRLF].(RFC-821, pg 34)

c. Multi line replies shall consist of a three digit code, followed immediately by a Hyphen "-", also known as a minus sign, followed by text. The last line shall begin with the same three digit code, followed by a [SP], some optional text, and terminated by a [CRLF]. (RFC-821, pg 50)

3.7.2.2.1 SMTP Reply Codes (RFC-821, pg 48)

The SMTP reply codes and their meaning shall be as shown below:

a. The first digit, most significant, of the reply code shall have one of the following values: (RFC-821, pg 48)

1. 1xx - Positive preliminary reply (RFC-821, pg 48)

   The SMTP does not have any commands that would allow this reply and it shall not be implemented.

2. 2xx - Positive completion reply (RFC-821, pg 48)

   The requested action has been successfully completed and a new command may be initiated.

3. 3xx - Positive intermediate reply (RFC-821, pg 48)

   The command has been accepted and the requested action is being held in abeyance pending receipt of another command from the sender SMTP specifies the required information.

4. 4xx - Transient negative completion reply (RFC-821, pg 48)

   The command has not been accepted and requested action did not take place. The error condition is temporary and the sender SMTP should reissue the command or return to the beginning of the command sequence.

5. 5xx - Permanent negative completion reply (RFC-821, pg 49)

   The command has not been accepted and the requrested action did not take place. The sender SMTP should not repeat the exact command or command sequence.

b. The second digit of the reply code shall indicate specific functional categories with one of the following values: (RFC-821, pg 49)

1. x0x - Syntax error (RFC-821, pg 49)

   These replies refer to syntactically correct commands that don't fit any functional category, are unimplemented or superfluous.

2. x1x - Information (RFC-821, pg 49)

   These replies are for requests for information such as status or help.

3. x2x - Connections (RFC-821, pg 49)

   These are replies referring to the transmission channel.

4. x3x - Unspecified (RFC-821, pg 49)

5. x4x - Unspecified (RFC-821, pg 49)

6. x5x - Mail system (RFC-821, pg 49)

   These replies indicate the status of the receiver mail system relative to the requested transfer or other mail system action.

c. The third digit, least significant, shall give a more precise meaning to each of the function categories specified by the first and second digit defined in above subsections a. and b. While the text in the following listed replies is only suggested, the first and second digit shown are mandatory and shall not be changed. (RFC-821, pg 49)

1. 211 System status, or system help ready (RFC-821, pg 36)

2. 214 Help message (RFC-821, pg 36) (Information on how to use the receiver or the meaning of a particular non-standard command.)

3. 220 (domain) Service ready (RFC-821, pg 36) (Where domain is the name of host receiving host)

4. 221 (domain) Service closing transmission channel (RFC-821, pg 36)

5. 250 Requested mail action okay and completed (RFC-821, pg 36)

6. 251 User not local, will forward to (forward-path) (RFC-821, pg 36) (Where forward-path is host name.)

7. 354 Start mail input, end with [CRLF].[CRLF] (RFC-821, pg 36)

8. 421 (domain) Service not available, closing transmission channel (RFC-821, pg 36) (This may be a reply to any command if the service knows it must shut down.)

9. 450 Requested mail action not taken, mailbox unavailable (RFC-821, pg 36) (For example, the mailbox may be busy.)

10. 451 Requested action aborted, local error in processing (RFC-821, pg 36)

11. 452 Requested action not taken, insufficient system storage (RFC-821, pg 36)

12. 500 Syntax error, command unrecognized (RFC-821, pg 36) (This may include errors such as command line too long.)

13. 501 Syntax error in parameters or arguments (RFC-821, pg 36)

14. 502 Command not implemented (RFC-821, pg 36)

15. 503 Bad sequence of commands (RFC-821, pg 36)

16. 504 Command parameter not implemented (RFC-821, pg 36)

17. 550 Requested action not taken, mailbox unavailable (RFC-821, pg 36) (For example, mailbox not found, no access.)

18. 551 User not local, please try (forward-path) (RFC-821, pg 36)

19. 552 Requested mail action aborted, exceeded storage allocation (RFC-821, pg 36)

20. 553 Requested action not taken, mailbox name not allowed (RFC-821, pg 36) (For example, mailbox syntax incorrect.)

21. 554 Transaction failed (RFC-821, pg 36)

3.7.2.2.2 Sequencing of Commands and Replies (RFC-821, pg 37)

The communication between the sender and the receiver shall be an alternating dialog implemented as follows.

a. The sender shall issue commands to the receiver and wait for a reply before sending further commands. (RFC-821, pg 37)

b. The receiver shall send one reply to the sender for each received command. (RFC-821, pg 37)

c. The specific COMMAND-REPLY sequences shall be as below (RFC-821, pg. 37):

NOTE: In subsections 1 through 15 below each command is listed and is followed by possible reply codes prefixed with "S" for success, "E" for error, "I" for intermediate, and "F" for failure.

1. Connection Establishment (RFC-821, pg 37)

   S: 220
   F: 421

2. HELO (RFC-821, pg 37)

   S: 250
   E: 500, 501, 504, 421

3. MAIL (RFC-821, pg 37)

   S: 250
   F: 552, 451, 452
   E: 500, 501, 421

4. RCPT (RFC-821, pg 38)

   S: 250, 251
   F: 550, 551, 552, 553, 450, 451, 452
   E: 500, 501, 503, 421

5. DATA (RFC-821, pg 38)

   I: 354 (followed by data, then)
   S: 250
   F: 552, 554, 451, 452

F: 451, 554
E: 500, 501, 503, 421

6. RSET (RFC-821, pg 38)

S: 250
E: 500, 501, 504, 421

7. SEND (RFC-821, pg 38)

E: 502 (This command not implemented)

8. SOML (RFC-821, pg 38)

E: 502 (This command not implemented)

9. SAML (RFC-821, pg 38)

E: 502 (This command not implemented)

10. VRFY (RFC-821, pg 38)

E: 502 (This command not implemented)

11. EXPN (RFC-821, pg 38)

E: 502 (This command not implemented)

12. HELP (RFC-821, pg 38)

S: 211, 214
E: 500, 501, 502, 504, 421

13. NOOP (RFC-821, pg 38)

    S: 250
    E: 500, 421

14. QUIT (RFC-821, pg 38)

    S: 221
    E: 500

15. TURN (RFC-821, pg 38)

    F: 502 (This command not implemented)

3.7.2.3 Transparency (RFC-821, pg 41)

To allow all user composed text to be transmitted transparently the following procedures shall be implemented.

a. Before sending a line of mail text the sender SMTP shall check the first character of the line for a period and, if found, shall insert one additional period at the beginning of the line. (RFC-821, pg 41)

b. When a line of mail text is received by the receiver SMTP, the receiver SMTP shall check the first character of the line for a period. If the first character is a period and there are no other characters in the line, then the receiver SMTP shall interpret the line as the end of mail. If the line contains additional characters, the receiver SMTP shall delete the first character (period) and interpret the line as mail data. (RFC-821, pg 41)

3.7.2.4 Sizes (RFC-821, pg 42)

The SMTP implementation shall be able to receive objects of at least the following sizes and shall not send objects larger than the following sizes. (RFC-821, pg 42)

a. USER NAME (RFC-821, pg 42)

   The maximum total length of the user name shall be 64 characters.

b. DOMAIN (RFC-821, pg 42)

   The maximum total length of a domain name or number shall be 64 characters.

c. PATH (RFC-821, pg 42)

   The maximum total length of a reverse-path or forward-path shall be 256 characters, including punctuation and element separators.

d. COMMAND LINE (RFC-821, pg 42)

   The maximum total length of a command line, including the command word and the [CRLF], shall be 512 characters.

e. REPLY LINE (RFC-821, pg 42)

   The maximum total length of a reply line, including the reply code and the [CRLF], shall be 512 characters.

f. TEXT LINE (RFC-821, pg 43)

   The maximum total length of a text line, including the [CRLF] but not including the duplicated period, shall be 1000 characters.

g. RECIPIENTS BUFFER (RFC-821, pg 43)

The maximum total number of recipients that must be buffered shall be 100.

3.7.3 External Interfaces (RFC-821)

The following subsections define the SMTP external interfaces. Interfaces are provided for direct user call of SMTP and for application calls to SMTP.

3.7.3.1 Application Program (RFC-821)

The Application Program interface shall be provided by SMTP to application programs written in FORTRAN or any language that can interface to FORTRAN (for example, SYMPL, CYBIL, COBOL). The SMTP Control Statement interface and the local host mail server shall use this interface to access the SMTP capabilities. In the following discussions, the term 'standard_character' string is used to refer to a FORTRAN character string or to an array of 60-bit words that contains left-justified characters, zero-filled and zero-byte terminated. A standard_character string uses the 6/12 display code representation, so the display code escape character will always be interpreted as such. If a standard_character string can contain n characters then the calling routine should allocate 2n 6-bit bytes (10 bytes per 60-bit word) for the worst case of all lower case characters. The term 'name_character' is used to refer to the character subset consisting of alphabetic (A-Z), numeric (0-9), minus sign (-), and period (.) characters.

3.7.3.1.1 SMTP_Accept_Mail (SMTPAM) Routine (RFC-821)

The SMTP Accept_Mail routine shall return to the application program a mail message received by SMTP. This function is used by local host mail servers to get SMTP mail and route it to the local host user. Only applications

that have connected to SMTP via the SMTP_Connect_Server routine may use this routine. The calling format shall be:

CALL SMTPAM (message_size, message, no_wait, reply_code, more_data)

message_size — Maximum number of 6-bit characters to be placed in the buffer. An integer number supplied by the application.

message — Buffer to receive the mail message. A standard_character string returned by SMTP containing up to message_size characters. The mail source (reverse_path) and mail destination (forward_path) can be obtained by the application from the message header.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately if no mail messages have arrived for processing. The application must then call SMTP_Accept_Mail again later to check again. If no_wait is supplied, it is important that the application check frequently or mail messages may be lost. If no_wait is false, the application is rolled out until a message arrives.

reply_code — Reply code from request. Integer value is returned by SMTP for completed requests regardless of success. Reply codes are defined in section 3.7.2.2.1. Reply code is zero or negative if no_wait is specified and no mail messages are available.

more_data — Indicates that the message buffer was not large enough to contain the entire mail message. Boolean value returned by SMTP. If true, a subsequent call to SMTP_Accept_Mail will return the next buffer of the message. When the end of the message is encountered (more_data false), a subsequent call to SMTP_Accept_Mail will return the data from the next mail message.

3.7.3.1.2 SMTP_Accept_Reply (SMTPAR) Routine (RFC-821)

The SMTP Accept_Reply routine shall return the response to the last request issued. This routine can also be used to wait for completion of requests that were previously issued with no_wait. The calling format shall be:

```
CALL SMTPAR  (reply, no_wait, reply_code, more_data)
```

reply — Textual reply from last SMTP request. A standard character string returned by SMTP containing up to 80 characters of reply text. The text for replies from a local CYBER SMTP is specified in section 3.7.2.2. This value is the string "Response pending," if reply_code is zero.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately if the previous request has not completed. The application must then call SMTP Accept_Reply again later to determine the reply.

reply_code — Reply code from previous request. Integer value is returned by SMTP for completed requests regardless of success. Reply codes are defined in section 3.7.2.2.1. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

more_data — Indicates the reply text is continued on another line (80 character array). Boolean value returned by SMTP. If true, a subsequent call to SMTP_Accept_Reply will return the next line of the message. When the last line is read (more_reply false), a subsequent call to SMTP_Accept_Reply will return the last line of the response.

3.7.3.1.3 SMTP_Connect (SMTPC) Routine (RFC-821)

The SMTP_Connect routine shall request the SMTP Protocol Interpreter to establish a connection with an SMTP residing on a remote host. An application may have only one SMTP connection established at once. The calling format shall be:

CALL SMTPC (reply_code)

reply_code — Reply from connection attempt. Integer value is returned by SMTP for all connection attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called.

3.7.3.1.4 SMTP_Connect_Server (SMTPCS) Routine (RFC-821)

The SMTP_Connect_Server routine shall request the SMTP Protocol Interpreter to send messages received that are destined for the local host. An

application may have only one SMTP connection established at once, but many independent servers may be connected simultaneously; in this case, a server is assigned to an available message if it is non-busy. The calling format shall be:

CALL SMTPCS (reply_code)

reply_code — Reply from connection attempt. Integer value is returned by SMTP for all connection attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called.

3.7.3.1.5 SMTP_Disconnect (FTPD) Routine (RFC-821)

The SMTP_Disconnect routine shall request the SMTP Protocol Interpreter to terminate a previously established connection. The calling format shall be:

CALL SMTPD (reply_code)

reply_code — Reply from disconnect attempt. Integer value is returned by SMTP for all attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called.

3.7.3.1.6 SMTP_Mail_Command (SMTPMC) Routine (RFC-821)

The SMTP_Mail_Command routine shall request the SMTP protocol interpreter to process a command, that is not directly supported by other SMTP calls. The

commands HELP, NOOP, RSET, and RSCT, as specified in section 3.7.2.1.5.1, are sent using this procedure. The calling format shall be:

CALL SMTPMC (command, no_wait, reply_code)

command	SMTP command to be issued. A standard_character string supplied by the application containing up to 512 characters. SMTP maintains state tables and will reject any command issued that is inappropriate for the current state, or for improperly formatted commands.
no_wait	Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call SMTP_Accept_Reply to subsequently determine the reply.
reply_code	Reply from command request. Integer value is returned by SMTP for all attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

3.7.3.1.7 SMTP_Mail_Data (SMTPMD) Routine (RFC-821)

The SMTP_Mail_Data routine shall request the SMTP protocol interpreter to send a message to the recipient previously specified in SMTPMR calls. The calling format shall be:

CALL SMTPMD (message_size, message, no_wait, reply_code)

message_size — Number of 6-bit characters in the message buffer.

message — Message data to be sent to previously defined recipients. A standard_character string supplied by the application containing any number of characters with no more than 1000 characters per line.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call SMTP_Accept_Reply to subsequently determine the reply.

reply_code — Reply from command request. Integer value is returned by SMTP for all attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

3.7.3.1.8 SMTP_Mail_File (SMTPMF) Routine (RFC-821)

The SMTP_Mail_File routine shall request the SMTP protocol interpreter to send the contents of a file as a mail message to the recipients specified in previous SMTPMR calls. The calling format shall be:

```
CALL SMTPMF (file_name, file_access, no_wait, reply_code)
```

file_name — Name of file containing data to be sent to previously defined recipients. A display code string supplied by the application containing up to 7 characters. If the file is not local to the job, or if file_parameters is specfied, SMTP protocol interpreter will attempt to acquire a permanent file by that name.

file_parameters	Permanent file access parameters. A display code string supplied by the application containing up to 80 characters of file access directives. The directives are identical to those supplied for NOS permanent file control statement requests (for example, PN=packname, UN=username) and should follow the same syntax. Both direct and indirect access files can be referenced; it is not necessary to specify which kind the file is. If parameter is zero, an empty string, or a blank string then the current default parameters for the local job are used, and a local file is accessed if present.
no_wait	Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call SMTP_Accept_Reply to subsequently determine the reply.
reply_code	Reply from command request. Integer value is returned by SMTP for all attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called. Reply code is zero or negative if no wait is specified and operation is currently in progress.

3.7.3.1.9 SMTP_Mail_Header (SMTPMH) Routine (RFC-821)

The SMTP_Mail_Header routine shall insert the requested Mail text header fields into a Mail header buffer to be sent later with a mail message. This routine permits an application to use other fields defined by the ARPANET

Intertext Mail Standard that are not directly maintained by SMTP. The calling format shall be:

CALL SMTPMH (header_field, no_wait, reply_code)

header_field — A mail header field consisting of a field-name and a field-body. A standard_character string containing up to 1000 characters. The field name can be any of the standard fields except 'data', 'from', 'to', and 'cc' which are all maintained directly by SMTP. Both the field-name and field-body are checked for adherance to the standard. If SMTP_Mail_Header is called more than once for a given field-name, the field-name will be entered only once in the message header with subsequent field-body entries following the first at the same indentation level.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call SMTP_Accept_Reply to subsequently determine the reply.

reply_code — Reply from command request. Integer value is returned by SMTP for all attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

3.7.3.1.10 SMTP_Mail_Recipient (SMTPMR) Routine (RFC-821)

The SMTP_Mail_Recipient routine shall identify a mail recipient for a subsequent message to be sent using the SMTP MD/SMTPMF requests. The calling format shall be:

join

CALL SMTPMR (distribution_class, forward_path, no_wait, reply_code)

distribution_class — Distribution class of the recipient. A standard_character string supplied by the application containing either the string 'CC',''TO', or 'BCC'. If 'CC' is supplied, the recipient forward_path is listed in the 'CC:' portion of the message header. If 'TO' is supplied, the forward_path is listed in the 'TO:' portion of the message header. If 'BCC' is supplied, the forward_path is listed in the 'BCC' portion of the message header.

forward_path — Forward path of recipient. A standard_character string supplied by the application containing up to 256 characters.

no_wait — Indicates no waiting for completion. Boolean value supplied by application. If true, the routine will return immediately after issuing the request. The application must then call SMTP_Accept_Reply to subsequently determine the reply.

reply_code — Reply from command request. Integer value is returned by SMTP for all attempts regardless of success. Reply codes are defined in section 3.7.2.2.1. If the application requires the message text, the SMTP_Accept_Reply routine must be called. Reply code is zero or negative if no_wait is specified and operation is currently in progress.

3.7.3.2 User Control Statement (RFC-821)

The Control Statement interface shall be provided by SMTP to terminal user jobs, batch jobs, and CYBER Control Language (CCL) procedures. If the control statement is invoked from a terminal, the program shall perform a dialogue with the user, displaying replies as they are received and sending commands as they are entered. All parameters are order-independent. The format of the control statement shall be:

```
SMTP    (I=command_file, L=output_file, NA, PC=prefix_char)
```

command_file — Name of local file containing SMTP commands. If omitted, file INPUT is assumed. If the file is assigned to a terminal, SMTP may perform an interactive dialogue with the user, depending on the output_file. If I=0, SMTP will take commands from subsequent lines of the current control statement file. SMTP commands are distinguished from control statements by the prefix character specified by the PC parameter. The prefix character is not required for commands in any file except the control statement file.

output_file — Name of file to receive SMTP messages. These messages consist of command replies from the SMTP protocol interpreter. If the file is assigned to a terminal and command_file is also assigned to a terminal, then SMTP will perform an interactive dialogue with the user. If L=0, output messages are discarded.

NA — No-abort parameter. If specified, SMTP will continue to read and process commands even if errors are encountered; normally, SMTP will abort the user if errors are encountered in a command file that is not assigned to a terminal. This parameter has no meaning for interactive users.

prefix_char — Command prefix character. This character must precede all commands which are embedded in the control statement file (in other words, if $I=0$) following the SMTP statement. If omitted, asterisk (*) is assumed.

The following command allows the user to specify header-fields in a message header.

INFO[SP][HEADER-FIELD] where:

HEADER-FIELD is a legal header-field as defined by the ARPANET Internet Text Message standard. All field names are permitted except DATE, FROM, TO, and CC which are maintained by SMTP.

The Receipt (RCPT) command has been extended to permit the string 'CC:' instead of the 'TO:' string. If supplied, SMTP will enter the forward-path in the CC field of the message header. Normally, it is entered in the TO field.

3.7.3.3 DDN/C170 Application Interface (RFC-821)

SMTP shall utilize the DDN/C170 Application Interface to access DDN. Section 3.8.3 defines the DDN/C170 Application Interface primitives. The following services of the Application Interface shall be used:

1) TCP Primitives (RFC-821)
2) Host name server translation and error reporting (RFC-821)
3) DDN error logging (RFC-821)

3.7.3.4 SMTP Peer Protocol (RFC-821)

SMTP shall implement the communicate with remote SMTP processes using the peer protocol specified in section 3.7.2.

3.7.3.5 Network Definition File (RFC-821)

SMTP shall define its NAM application parameters via NDL configuration directives. The directives used are defined in section 3.7.8.

3.6.3.6 Operator Interface

SMTP shall allow the operator to exercise some control over which messafes ae written to the log file. The following are the K-display HOP commands used to control message wiring to the log file.

K.DE = SMTP	Write error messages only to the log file
K.DB = SMTP	Write debug and error messages to the log file
K.RS = SMTP	Write statistics, debug, and error messages to the log file

Error messages are always written to the log file regardless of any operator intervention.

If the application has been loaded using NETIOD, the K.RS command will also restart AIP statistical gathering.

3.7.4 Internal Interfaces (RFC-821)

The following subsection describe the interfaces between the major components of CYBER SMTP. The internal interfaces identified are:

Control Statement to Application Interface

3.7.4.1 Control Statement to Application Interface (RFC-821)

The Control Statement to Application Interface is implemented by the AI routines. These routines are used by SMTP C/S to issue SMTP requests on behalf of an interactive terminal user or NOS batch job. This interface is defined in subsection 3.7.3.1.

3.7.5 Errors and Error Recovery (RFC-821)

3.7.5.1 Size Errors (RFC-821, pg 43)

The following error reply codes shall be used for errors in object size specified in 3.7.2.4a thru 3.7.2.4g.

a. 500 - line too long (RFC-821, pg 43)

b. 501 - path too long (RFC-821, pg 43)

c. 552 - too many recipients (RFC-821, pg 43)

d. 552 - too much mail data (RFC-821, pg 43)

3.7.6 Data Structures

Mail data shall conform to RFC-822, Standard for ARPA Internet Text Messages. (RFC-821, pg 5)

NOTE: In the following subsections, references are made to formal definitions of syntax contained in RFC-822. These references, although correct, will not provide the total definition of all syntactical entities and the reader, reviewer, developer, tester, etc. is referenced to Appendix L which is an extension of this ERS section 3.7. Appendix L contains the complete and formal syntax definitions for SMTP as reproduced from RFC-822. In the event of conflict between this ERS and the formal syntax described in Appendix L, Appendix L shall be the document of precedence.

Appendix L - Pages 3, 4, 9, 10, 11, 17, 18, 19, 26, 27, 44 through 47.

3.7.6.1 ARPA Internet Text Messages (RFC-822, pg 1)

The ARPA Internet Text Message Standard specifies the syntax for text messages sent among computer users within the framework of electronic mail. It defines a message as consisting of some information in a rigidly defined format followed by additional information in an unspecified format.

The rigidly defined portion of a message is referred to as a header field which consists of a field name and a field body. The field name consists of ASCII characters and is separated from the field body by a colon ":". Field bodies are of two types, termed structured field bodies and unstructured field bodies, with the difference being that structured field bodies are defined by a formal syntax, and unstructured field bodies consist of simply text.

The information in the message without a specified format is the main part of the message and can be thought of as the message body. The message body consists of all text from the null line after the last header field to the end of mail indicator which is a line containing only a period as the first and only character.

Figure 3.7-3, SMTP Implemented Message Standard, illustrates the above discussion and the requirements for implementation as stated in subsection 3.7.6.1.1 below.

The figure is to be interpreted as being read from the top: Message to Header-Field and then Field-Name and/or Field-Body, then from the top to the right when a Null-Line is detected and down to Message-Body.

Following the above interpretation, the figure illustrates that a mail Message consists of a Header-Field which contains a Field-Name and Field-Body made up of Structured-Fields and Unstructured-Fields which is followed by information contained in the mail Message-Body.

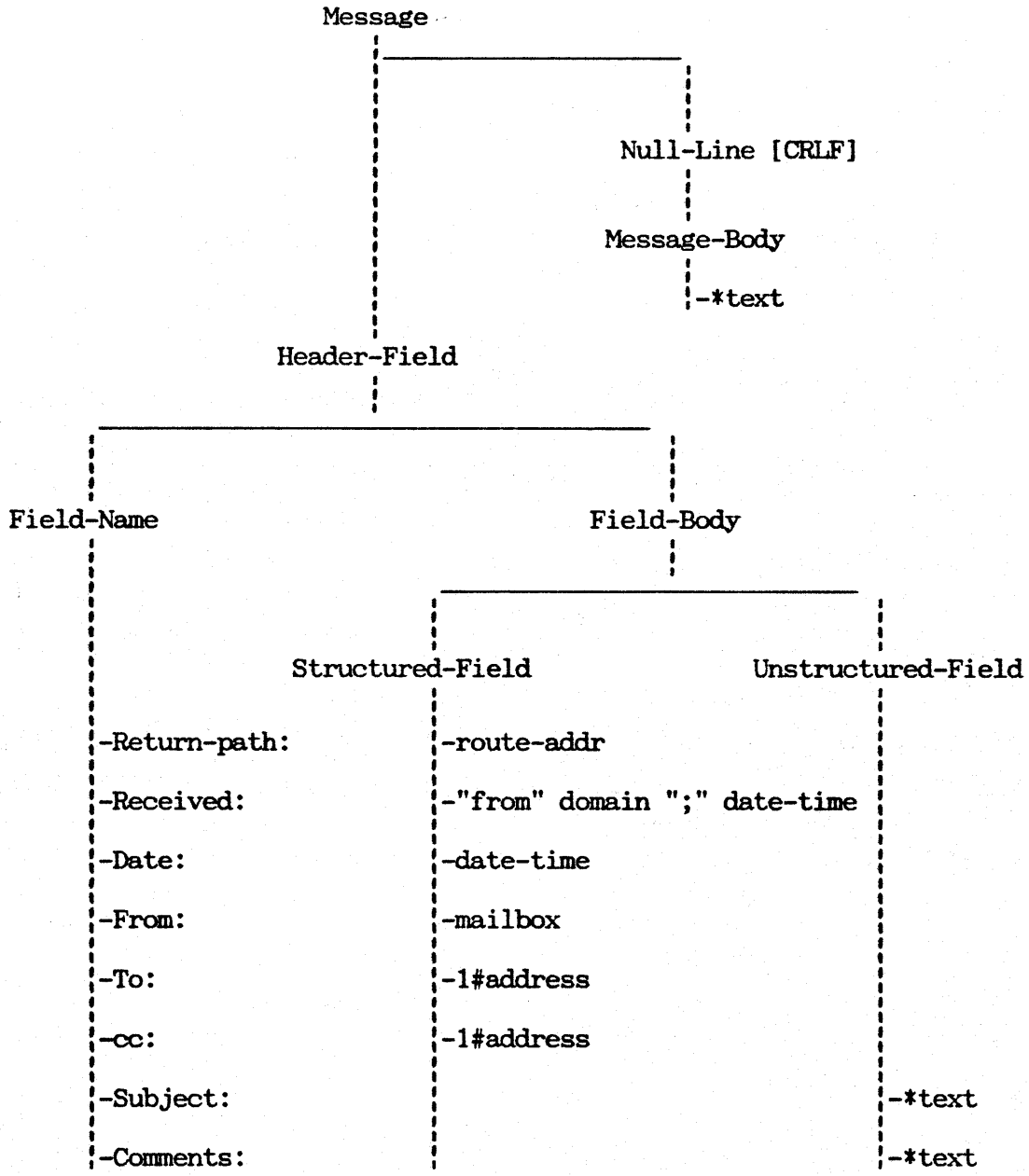


Figure 3.7-3. SMTP Implemented Message Standard

With all of the above in mind, the following is an example of a mail Message that would follow the DATA COMMAND in 3.7.2.1.5.1d:

Date: 20 Jan 85 09:00:00
From: R J Yuma <rjyuma@CDCNOS.SSDF>
Subject: Preliminary Design Review
To: rjschal@CDCNOS.SSDF
[CRLF] this is null line
Ron:
The DDN PDR is scheduled for 29 Jan 85. Please be prepared.
Russ
. this terminates the message body

3.7.6.1.1 Header Field Definitions (RFC-822, pg 9)

a. The formal definition of HEADER FIELDS shall be as shown on page 9 of RFC-822. (RFC-822, pg 9)

b. The header field shall consist of a field-name followed by a colon ":", followed by a field-body and terminated by a carriage return/line feed. (RFC-822, pg 6)

c. The field-name shall be composed of printable ASCII characters with decimal values between 33 and 126, except for the colon ":", decimal value 58. (RFC-822, pg 6)

d. The field-body shall be composed of a structured field-body or an unstructured field-body as defined below. (RFC-822, pg 6)

1. Structured field body (RFC-822, pg 6)

Any field-body that is defined as other than simple text shall be processed as a structured field according to an internal syntax. (RFC-822, pg 6)

2. Unstructured field body (RFC-822, pg 6)

   Any field-body composed of simply text shall be defined as unstructured and processed as a string of text. (RFC-822, pg 6)

e. The field-names that shall be implemented are shown below. (RFC-821, pg. 20

   1. Return-path: (RFC-822, pg 20)

      The structured field-body for this field-name shall be added by the receiving SMTP that delivers the message to its recipient. The field is intended to contain definitive information about the address and route back to the message's originator.

   2. Received: (RFC-822, pg 20)

      The structured field-body for this field-name shall be added by the receiving SMTP that accepts a mail message for delivery to its recipient.

   3. Date: (RFC-822, pg 17,18)

      The structured field-body for this field-name shall be added by the sending SMTP and shall contain the date and time that the message was sent.

   4. From: (RFC-822, pg 21)

      The structured field-body for this field-name shall be created by the sending SMTP. It shall contain an authenticated machine address that is machine usable that indicates the person, system, or process creating the message and it shall not contain lists.

5. To: (RFC-822, pg 23)

   The structured field-body for this field-name shall contain the identity of the primary recipients of the message and shall be generated by the sender SMTP.

   NOTE: The sender SMTP shall allow "Postmaster" in upper and/or lower case as a legal part of the local part of a mailbox (see note in f4 below). (RFC-822, pg 33)

6. CC: (RFC-822, pg 23)

   The structured field-body for this field-namd shall contain the identity of the secondary recipients of the message and shall be generated by the sender SMTP.

7. Subject: (RFC-822, pg 24)

   The unstructured field-body for this field-name is intended to provide a summary or indicate the nature of the message and shall be generated by the sender SMTP.

8. Comments: (RFC-822, pg 24)

   The unstructured field-body for this field-name is intended to allow adding comments onto the message without disturbing the contents of the message body and shall be generated by the sender SMTP.

f. The corresponding field-bodies for the above field-names shall be as shown below. (RFC-822, pg. 17)

   1. route-addr (RFC-822, pg 17)

2. "from" domain "by" domain ";" date-time (RFC-822, pg 17)

3. date-time (RFC-822, pg 18)

4. mailbox (RFC-822, pg 18)

5. 1#address (RFC-822, pg 18)

   NOTE: The receiver SMTP shall recognize "Postmaster" as a reserved mailbox where "Postmaster" is the local part of the address and shall be recognized in either upper and/or lower case. Mail sent to this address shall be routed to a person responsible for the site's mail system. (RFC-822, pg 33)

6. 1#address (RFC-822, pg 18)

7. *text (RFC-822, pg 18)

8. *text (RFC-822, pg 18)

3.7.7 Equipment Configuration

SMTP has no special equipment configuration requirements.

3.7.8 Installation (RFC-821)

The following subsections define the installation procedures and parameters for SMTP.

3.7.8.1 SMTP Installation Procedure (RFC-821)

SMTP is installed using the DDNSMTP build procedure and build verification is performed using the VDSMTP procedure. SMTP should be treated as a Group 3

job as defined in the NOS Installation Handbook. It depends upon the DDN Supervisor Installation defined in subsection 3.8.8. The SMTP product PL is maintained in MODIFY format. The following files are created or updated by the build procedure: JOBSMTS, HELPLIB(updated), DDNLIB(updated).

3.7.8.2 SMTP Installation Parameters (RFC-821)

The following installation parameters can be tailored for specific sites. They are located in common deck COMVIPR.

IPDLCS — Default Local Character Set. This is the default character set for commands, responses and files at the local CYBER host. Possible values are ASCII, ASCII8, and DIS as defined by the FCOPY statement. The default value is ASCII.

IPMCTO — SMTP Command Timeout. This is the maximum time in seconds that SMTP will wait for a remote SMTP to respond to a command. After this period, the connection will be aborted. The default is 600.

IPMBUF — Maximum Buffer Size. This is the buffer size used for mail transfers. The minimum size is the NAM downline block size (255 words default). The default value is 1024B.

IPMNCP — SMTP NOOP Command Period. This is the period in seconds at which SMTP will issue NOOP commands to a remote SMTP during wait periods to prevent timeout.

IPRTRY — Retry Limit. This is the maximum retry limit for errors before SMTP terminates the connection.

The following modify symbols can be specified using the *DEFINE directive to influence the SMTP build process:

DEBUG If defined, SMTP will generate AIP trace messages.

IMS If defined, the SMTP listing will contain Internal Maintainance information.

STATS If defined, SMTP will generate AIP statistics messages.

3.7.8.3 Network Startup Master File (RFC-821)

The following directives must be placed in the Network Startup Master File if automatic initiation of SMTP is required.

```
PARAM(SMTPMS=n)
```

Where n is the number of SMTP mail servers to initiate.

```
JOB(JOBSMTS,SMTS)
```

3.7.8.4 Network Definition File Directives (RFC-821)

The following table lists the DDN applications and their configuration parameters:

Name	Description	Number of Copies (MXCOPYS)	Request Startable	Privileged
SMPI	SMTP Protocol Interpreter	1	yes	yes
SMPS	SMTP Mail Server	1	no	yes
SMTP	SMTP Control Statement	any(5)	no	no
SMPA	SMTP Mail Application	any(1)	no	no

The values for MXCOPY can be adjusted by the site to control SMTP usage. A small number can mean that SMTP users will have to wait for resources while a large number can tie up TCP and NAM connections and machine resources, affecting other users of the system.

Additionally, SMTP application users must provide OUTCALL directives to SMTP for their applications.

3.7.8.5 SMTP Transmittal

No other transmittal can depend upon this transmittal.

a. SMTP Program Library (SMPnnnn). This is an UPDATE formatted program library containing the SMTP application interface routines, Protocol Interpreter, SMTP Control Statement. The format of this program library is described later. The following object is generated from this PL:

Name	Type	Location	Description
SMPI	Program	System	SMTP Protocol Interpreter
SMTP	Program	System	SMTP Control Statement
DDNLIB	Library	System	SMTP Additions to DDN Interface Library
NAMSTRT	Procedures	un=NETOPS	NAM Startup Proc. Update for SMPI (JOBSMPI)
HELPLIB	CCL Library	un=LIBRARY	SMTP Help Procedures Update to HELPLIB (SMTP/SMPPEER)

b. SMTP Installation procedure (SMTP). This procedure follows DECKOPL conventions to install SMTP.

c. SMTP Installation Verification procedure (VSMTP). This procedure is to be executed on a newly-built system to verify that SMTP has been properly installed. It does not verify that SMTP actually works. It follows DECKOPL verfication conventions.

3.7.8.6 PL Structure

a. Where deck lists are described as 'organized', it means that the decks are arranged in a SIMPLE manner logical with the structure of the program. For example, the main program followed by all primary routines in alphabetical order followed by all minor routines in alphabetical order. If no structure can be identified, 'organized' degenerates to 'alphabetized', never to 'random' or 'complex'.

b. SMTP PL Structure.

```
INFO
HISTORY
[alphabetized common decks]
-LIBSYMP-  [contains *CWEOR]
[SYMPL library routines, alphabetized]
-LIBCOMP-  [contains *CWEOR]

[COMPASS library routines, alphabetized]
-SYMPL-    [contains *CWEOR]
-PISYMP-   [empty deck]
[SMPI SYMPL routines, organized]
-CSSYMP-   [empty deck]
[SMTP CS SYMPL routines, organized]
-COMPASS-  [contains *CWEOR]
-PICOMP-   [empty deck]
[SMPI COMPASS routines, alphabetized]
-CSCOMP-   [empty deck]
[SMTP  CS COMPASS routines, alphabetized]
-NAMSTRT-  [contains *CWEOR]
JOBSMPI
-HELPLIB-  [contains *CWEOR]
SMTP
-HEOR1-    [contains *CWEOR]
SMPPEER
-ENDPL-    [empty deck]
```

NOTE: All non-library SYMPL/COMPASS decks are compiled into a large library. The SMPI and SMTP main programs are separately loaded using a LIBLOAD from that library.

c. Inter-Component Common Decks

DDN System-Wide (defined in DDNTEXT)

COMVDDN	DDN General Runtime Interface Definitions
COMVDIP	DDN System Installation Parameters
COMVFET	CIO FET Definitions
COMVFIT	CRM FIT Definitions
COMVIPC	IP Interface Definitions
COMVNCT	DDN NCT Definitions
COMVSYS	CYBER System Definitions
COMVTCP	TCP Interface Definitions
COMVTEL	TELNET Interface Definitions

SMTP

COMVAPC	SMTP Application to Protocol Interpreter Interface
COMVIPM	SMTP Installation Parameters
COMVSMP	SMTP Definitions

3.8 DDN Gateway (DDN G/W)

3.8.1 DDN G/W Abstract

The purpose of DDN Gateway is to provide general DDN services to CYBER 170 applications without requiring application knowledge of the CDCNET implementation of those services.

The Gateway will provide a DDN Name Server function, a common DDN message logging function, and TCP/IP/TELNET primitives and indications.

3.8.2 DDN G/W Description

Figure 3.8-1 depicts the DDN Gateway functional relationships. The gateway is divided into three components: 1) The DDN Supervisor, 2) the DDN/C170 Gateway, and 3) the DDN Application Interface.

The DDN Supervisor provides the Host Name server and message logging functions. A NAM application residing in either the 170 or an MDI must establish a connection with the Supervisor, which resides in the 170, and can then request DDN Name-to-Address translations or message logging. The name translation facility is required by FTP and SMTP to translate DDN domain names supplied by the user into internet addresses required by TCP/IP. Common message logging is provided so that DDN-specific errors detected by independent applications (for example, FTP, SMTP) are logged in a single location for efficient problem analysis and resolution.

The DDN/C170 Gateway provides C170 application access to the TCP/IP/TELNET services residing in CDCNET. A NAM application residing in the 170 establishes a connection with the DDN/C170 Gateway residing in the CDCNET and can then issue TCP/IP/TELNET requests and receive their indications. The Gateway is required due to differences in architecture between the C170 Network Products used by applications and CDCNET where TCP/IP/TELNET is implemented. Primitives and indications are passed over the NAM connection between the applications and the Gateway. The Gateway in turn issues the primitives to TCP/IP/TELNET, or receives the indications.

The DDN Application Interface (DDN AI) provides a standard application interface to the services provided by DDN Supervisor and DDN/C170 Gateway. The AI is a set of routines that can be called by application programs that handle the NAM connection and data interfaces required by Supervisor and C170 Gateway. These routines consists of a DDN registration function, a name server function, a message logging function, and all the TCP/IP/TELNET primitives and indications.

The various G/W components communicate via NAM connections that are depicted in Figure 3.8-1 and summarized below:

1) Application to DDN Supervisor. This connection is initiated and terminated by the AI. One connection exists for each instance of the Application I/F, which is resident in each application that uses DDN services, including SMTP and FTP applications. Any number of connections are supported, up to the maximum configured by the network administrator.

2) Application to DDN/C170 Gateway. The first connection for any given NAM application is initiated by the application, while subsequent connections can be initiated by either the application or the Gateway. Connections can be terminated by either the application or the Gateway. One connection exists for each and every TELNET, TCP, or IP connection established with an application.

3.8.2.1 Gateway Connection Registration

A Gateway Connection registration function shall be provided to a CYBER 170 application that shall provide access to DDN services. DDN service access by unregistered applications shall be rejected. The registration service shall require the user to specify the NAM application name and the type of services to be used. The service types include direct NAM interface, host protocols (for example, TCP/IP/TELNET), or application protocols (for example, FTP/SMTP). Any successful registration request shall establish a connection with the DDN Supervisor. A successful registration request for host protocol services shall establish a connection to the DDN/C170 Gateway.

A function shall also be provided that allows an application to delete a previously established registration. If the registration is deleted, any outstanding Gateway/NAM connections with the application shall be unilaterally terminated.

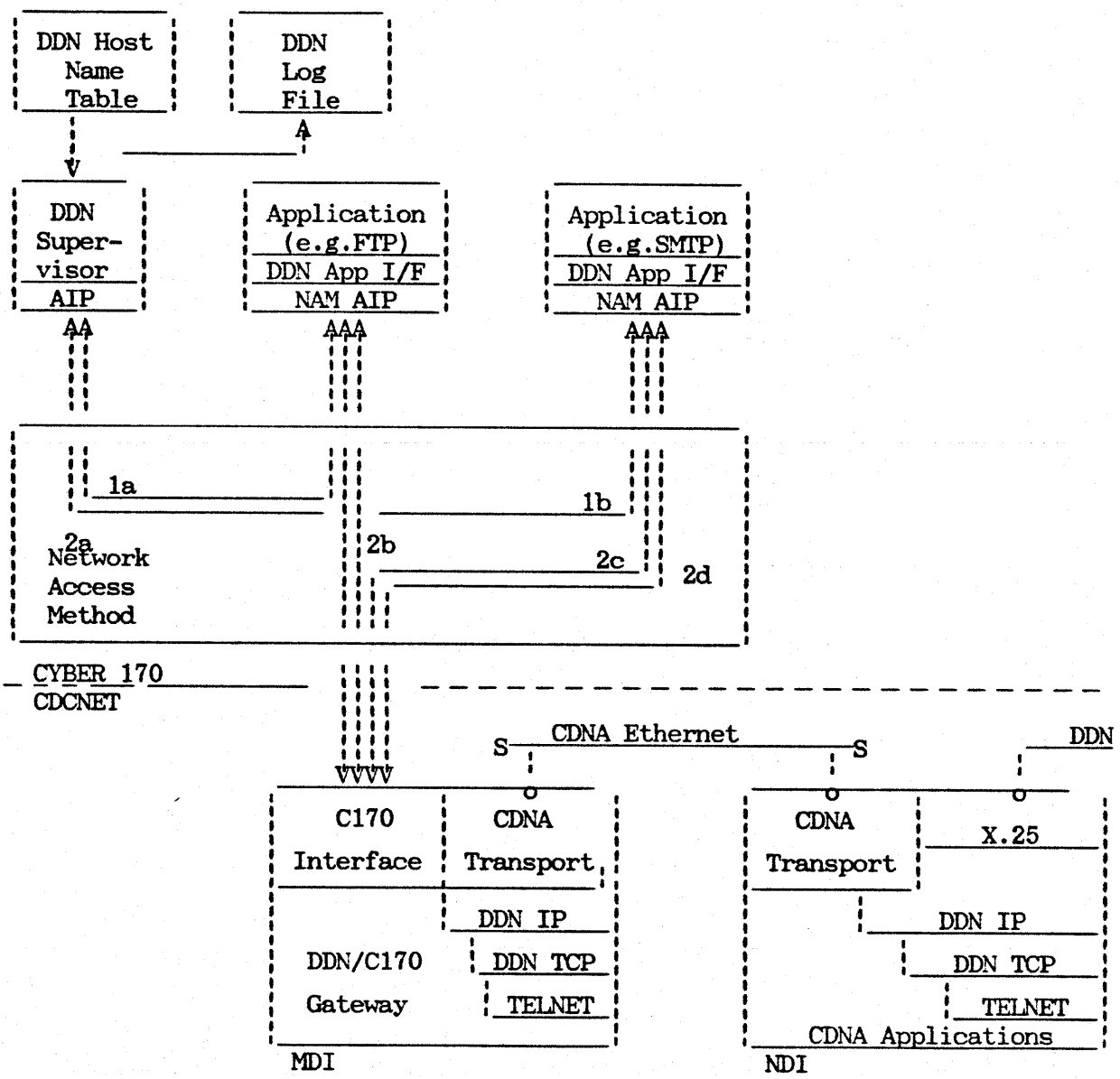


Figure 3.8-1. DDN Gateway Functional Relationships

3.8.2.2 DDN Name Translation

A DDN Domain Name translation function shall be provided by the Gateway CPCI to registered CYBER 170 applications. The translation function shall accept a DDN domain name as defined in RFC-810. If an entry for the name is located in the table, an internet address as defined in RFC-923 shall be returned. If no entry is found, a condition code shall be returned. The function shall also provide a protocol service verification if requested by the user.

The translation function shall use the DDN host name table format as specified in RFC-810.

3.8.2.3 DDN Message Logging

A DDN Message logging function shall be provided by the Gateway CPCI to registered CYBER 170 applications. The logging function shall accept a message from an application and shall log the message in a common file along with the time and application name.

3.8.2.4 TELNET Primitives and Indications

The Gateway CPCI shall provide CDCNET TELNET primitive and indication services to CYBER 170 application programs. These primitives and indications shall be provided as part of the TCP interface in the form of an option selecting TELNET protocol translation as specified in the TELNET ERS.

3.8.2.5 TCP Primitives and Indications

The Gateway CPCI shall provide CDCNET TCP primitive and indication services to CYBER 170 application programs. The services provided shall be identical to those specified in the TCP ERS.

If an application issues a TCP open service request, the DDN/C170 Gateway shall initiate a NAM connection with the application which is used as the dedicated data path for that TCP connection. If the TCP connection request fails or the connection is terminated, the NAM connection shall be terminated.

If an application has previously issued a successful TCP passive open service request, and a TCP connection request indication is received by the DDN/C170 G/W, the Gateway shall establish a NAM connection with the application. When the connection is terminated, or fails, G/W shall terminated the NAM connection.

The DDN/C170 gateway shall maintain a one-to-one correspondence between TCP connections and NAM connections.

3.8.2.6 IP Primitives and Indications

The Gateway CPCI shall provide CDCNET IP primitive and indication services to CYBER 170 application programs. The services provided shall be identical to those specified in the IP ERS.

If an application issues an IP open service request, the DDN/C170 Gateway shall initiate a NAM connection with the application which is used as the dedicated data path for that IP connection. If the IP service request fails, or the connection is terminated, the NAM connection shall be terminated.

The DDN/C170 gateway shall maintain a one-to-one correspondence between IP service access points and NAM connections.

3.8.3 External Interfaces

The following subsections define the DDN Gateway CPCI external interfaces.

3.8.3.1 Application Program

The application program interface shall be provided by the Gateway CPCI to application programs written in FORTRAN or any language that can interface to FORTRAN (for example, SYMPL, CYBIL, COBOL).

3.8.3.1.1 DDN General Service Requests

The following routines provide general DDN services including registration, name translation, and message logging.

3.8.3.1.1.1 DDN_Register_Application (DDNRA) Routine

The DDN_Register_Application routine shall register an application with DDN. This routine must be called before any other routine defined in section 3.8.3.1 can be used. This routine must also be called by any application protocol user (for example, user of FTP or SMTP) if that user is also a NAM application.

CALL DDNRA (application_name, NAM_direct_user, number_connections, reply)

application_name — Name of application. This is the name given to NAM and specified in APPL/INCALL/OUTCALL directives in the NDL. Upper-case display code string from 1-7 characters.

NAM_direct_user	Highest NAM connection group in use by the application. This value must be zero if the application does not make direct calls to NAM; this value must be specfied if the application does make NAM direct calls. DDN will use the next highest connection group for its NAM connections. The application must take care not to accept or send messages over that group. Use of NAM direct access with DDN is discouraged.
number_connections	Integer value: maximum number of connections to support.
reply	Integer reply code from registration request. Possible values are: Application_Registered Previously_Registered Privileged_Access Network_Unavailable

3.8.3.1.1.2 DDN_Delete_Registration (DDNDR) Routine

The DDN_Delete_Registration routine shall delete a previously established registration. This routine must be called when an application no longer requires DDN services.

```
CALL DDNDR    ()
```

3.8.3.1.1.3 DDN_Translate_Name (DDNTN) Routine

The DDN_Translate_Name routine shall search for the specified name in the host name table and return the associated entry if the name is found.

CALL DDNTN (name, required_protocol, entry_type, address_list, address_size, name_list, name_size, system, protocol_list, protocol_size, comment, comment_size, entry, entry_size, reply, no_wait)

name — Name to be translated. A standard_character string supplied by the application containing up to 24 characters.

required_protocol — Protocol required. This is one of the protocols listed in RFC-810. If the protocol does not appear in the protocol list of the entry, an indication will be returned. A standard_character string supplied by the application containing up to 24 characters. If zero or null, no protocol matching is performed.

entry_type — Type of entry. A character string returned by G/W containing up to 7 characters. Possible values are NET, GATEWAY, or HOST.

address_list — List of internet addresses. An array of character strings returned by G/W containing internet addresses associated with the name. Each string is up to 20 characters long.

address_size — Length of address array. An integer supplied by the application specifying the size of the array. If less space is provided than there are entries, the list is truncated.

name_list — List of alternate names. An array of character strings returned by G/W containing alternate names associated with the name. Each string is up to 24 characters long.

name_size	Length of name array. An integer supplied by the application specifying the size of the array. If less space is provided than there are entries, the list is truncated.
system	Name of Operating System. A character string returned by G/W containing up to 24 characters.
protocol_list	List of supported protocols. An array of character strings returned by G/W containing protocols supported. Each string is up to 20 characters long.
protocol_size	Length of protocol array. An integer supplied by the application specifying the size of the array. If less space is provided than there are entries, the list is truncated.
comment	Comment string. A standard_character string returned by G/W containing the entry comment.
comment_size	Length of comment string. An integer supplied by the application specifying the size of the comment character string. If the supplied string is smaller than the entry string, the string is truncated.
entry	Full host name table entry. A standard_character string returned by G/W containing the exact full entry.
entry_size	Length of entry string. An integer supplied by the application specifying the size of the entry character string. If the supplied string is smaller than the entry, the string is truncated.

reply — Integer reply code from translation request. Possible values are:
- Name_Translated
- Protocol_not_Supported
- Application_not_Registered

no_wait — Boolean value, if true, routine will return immediately after issuing request. The application must then call the appropriate accept_indication routine (TCPAI, IPAI) to subsequently determine the status of the request.

3.8.3.1.1.4 DDN_Log_Message (DDNLM) Routine

The DDN_Log_Message routine shall place the specified message in a common log file with the time and application name.

CALL DDNLM (message, message_code, destination, level, source, reply, no_wait)

message — Message to be logged. An upper-case display code character string supplied by the application containing up to 80 characters and zero_byte terminated.

message_code — Numeric message identifier. An integer in the range 0-9999 that uniquely identifies the message for this application.

destination

level

source

reply — Integer reply code from logging request. Possible values are:
- Message_Logged
- Privileged_Access
- Application_not_Registered

no_wait — Boolean value, if true, will cause routine to return immediately after issuing request, with reply set to message_logged regardless of the success of log request.

Error messages are always written to the log file. Debug and statistical messages are controlled at installation time or via operator intervention using the NAM K-display HOP commands K.DE, K.DB, and K.RS.
K.DE 'activates' error messages only (that is, deactivates debug and statistics). K.DB activates error and debug ('deactivating' statistics), and K.RS activates error, debug, and statistics.

3.8.3.1.2 TCP Primitives

3.8.3.1.2.1 TCP_Open_Sap (TCPOS) Routine

The TCP_Open_Sap routine is used to open a Service Access Point (SAP) with the TCP module. It is at this time that the application registers the

addresses of its service routines, and receives the addresses of the TCP request routines. The format shall be:

CALL TCPOS (user_sapid, tcp_sapid, status, no_wait)

user_sapid — This is a record that contains the user SAPID and a list of user defined routines. The SAPID is an integer value whose range is -2**31 to 2**31-1.

tcp_sapid — This is the record returned that contains the TCP SAPID and a list of the TCP service routines. The tcp_sapid is an integer value whose range is -2**31 to 2**31-1.

status — This is the status returned as a result of the open request. The following integer values may be returned:

tcp_sap_open
tcp_sap_unavailable

no_wait — Boolean value, if true, causes routine to return immediately. Application must then call TCPAI to subsequently determine results.

3.8.3.1.2.2 TCP_Close_Sap (TCPCS) Routine

The TCP_Close_Sap routine closes a currently open SAP. All connections through the SAP are aborted before the SAP is closed. The calling format shall be:

CALL TCPCS (tcp_sapid, status, no_wait)

tcp_sapid — This is the SAPID returned by TCP in the open SAP request. It is an integer value whose range is -2**31 to 2**31-1.

status — This is the status returned as a result of the close request. The following integer values may be returned:

tcp_sap_closing
tcp_sap_not_open

no_wait — Boolean value, if true, causes routine to return immediately after issuing request. Application must then call TCPAI to subsequently determine results of request.

3.8.3.1.2.3 TCP_Active_Connect (TCPAC) Routines

The TCP_Active_Connect will attempt to connect to a peer at the specified destination. The status of the request will be returned. The calling format shall be:

CALL TCPAC (tcp_sapid, user_cepid, tcp_cepid, source, destination, allocation, ip_header, ip_options, ulp_timeout, push_flag, urgent_flag, data, data_length, status, no_wait)

tcp_sapid — This is the SAPID returned by TCP in the open SAP request. The SAPID is an integer value whose range -2**31 to 2**31-1.

user_cepid — This is the Connection End Point Identifier (CEPID) which will be provided by TCP with all data passed to the application. The CEPID is an integer value whose range is -2**31 to 2**31-1.

tcp_cepid — This is the CEPID created by the TCP that is returned. It must be specified on all subsequent requests. The CEPID is an integer value whose range is -2**31 to 2**31-1.

source — This parameter specifies the internet address and port address of the datagram source. The internet address is optional, and in the multihost system, will specifiy the network solution used to send the datagram. There are a number of well known ports, an application not owning such a port should leave this field unspecified. All unspecified fields will be filled in and returned by TCP.

The internet address (ip_address) is a 32 bit value whose range is -2**31 to 2**31-1. The port address is a 16 bit value whose range is 0 to 2**15-1. There is a boolean value appended to the addresses that indicates whether a port address is specified. This value is set as follows: 0 = TRUE, 1 = FALSE.

destination — This parameter specifies the internet address and port address of the destination application.

The internet address (ip_address) is a 32 bit value whose range is -2**31 to 2**31-1. The port address is a 16 bit value whose range is 0 to 2**15-1. There is a boolean value appended to the addresses that indicates whether a port address is specified. This value is set as follows: 0 = TRUE, 1 = FALSE.

allocation — Integer number of words the ULP will accept. Range is 0 - 2*16.

ip_header — This is a 19-word record that contains the IP header for the data gram. This record is used by the ULP to pass header information to IP. The fields to be completed by the ULP are specified below:

Word	Field Description	Range	ULP Supplied
0	version	0..15	
1	ihl	0..15	
2	precedence	0..7	yes
3	delay	Boolean	yes
4	throughput	Boolean	yes
5	reliability	Boolean	yes
6	unused_flag_1	Boolean	zero
7	unused_flag_2	Boolean	zero
8	total_length	0..0FFFF(16)	
9	identification	0..0FFFF(16)	yes
10	unused_flag_3	Boolean	zero
11	dont_frag	Boolean	yes
12	more_frags	Boolean	
13	frag_offset	0..01FFF(16)	
14	time_to_live	0..255	yes
15	protocol	0..255	yes

Word	Field Description	Range	ULP Supplied
16	checksum	0..FFFF(16)	
17	source	-80 000 000..7FFFFFFF(16)	
18	destination	-80 000 000..7FFFFFFF(16)	

ip_options

This is a 47-word record that contains the IP options for this connection. It consists of four subrecords for security, stream id, routing, and timing. The format of this record shall be as follows:

Word	Field Description	Range
0-5	ip_sec_rec	-
0	zero	0..127
1	in_use	Boolean
2	security	0..0FFFF(16)
3	compartments	0..0FFFF(16)
4	handling	0..0FFFF(16)
5	tcc	0..0FFFF(16)
6-8	ip_streamid_rec	-
6	zero	0..127
7	in_use	Boolean
8	streamid	0..0FFFF(16)
9-22	ip_routing_rec	-
9	zero	0.127
10	in_use	Boolean
11	rtype	0..255
12	index	0..255
13	count	0..255
14-22	ip_list [array 0..8]	-80 000 000..7FFFFFFF(16)
23-46	ip_time stamp	-
23	zero	0..127
24	in_use	Boolean

Word	Field Description	Range
25	ttype	0..255
26	index	0..255
27	count	0..255
28	overflow	0..255
29-46	sum_array [array 0..8]	ip_double_word
29-46	bg_array [array 0..3]	ip_full_stamp

ulp_timeout

push_flag

urgent_flag

data — This is a pointer to the buffer containing data to be sent with the connect request. If there is no data to be sent with the request, then the pointer should be NIL. The TCP module assumes ownership of one copy of this buffer.

data_length — Integer count of number of words in data buffer.

status — This is the status returned as a result of the active connection request. The following integer values may returned:

tcp_connecting
tcp_sap_not_open
tcp_illegal_source
tcp_illegal_destination
tcp_illegal_option
tcp_connection_inuse

no_wait — Boolean value, when true, causes routine to return immediately after issuing request.

3.8.3.1.2.4 TCP_Passive_Connect (TCPPC) Routine

The TCP_Passive_Connect routine is called by the application to inform the TCP module that it is willing to accept connect indications from a peer. The peers address may be specified completely, partially, or not at all. When an active connect request is received by the TCP module, the application will be given an indication. The application may accept or reject the connection and the passive connect will remain in effect. The calling format shall be:

CALL TCPPC (tcp_sapid, user_cepid, tcp_cepid, source, destination, allocation, ip_header, ip_options, ulp_timeout, push_flag, status, no_wait)

tcp_sapid — (tcp_sapid, user_cepid, tcp_cepid, source, destination, allocation, ip_header, ip_options, ulp_timeout, push_flag, status, no_wait)

user_cepid — (tcp_sapid, user_cepid, tcp_cepid, source, destination, allocation, ip_header, ip_options, ulp_timeout, push_flag, status, no_wait)

copy # 3.8.3.1.2.3 after "CALL TCPAC (.... ...., no_wait)

source (tcp_sapid, user_cepid, tcp_cepid, source, destination, allocation,ip_header, ip_options, ulp_timeout, push_flag, status, no_wait)

destination (tcp_sapid, user_cepid, tcp_cepid, source, destination, allocation,ip_header, ip_options, ulp_timeout, push_flag, status, no_wait)

options The option parameter specifies a record that contains the option specifications for this connection. The following is a list of the options and their range values:

3.8.3.1.2.5 TCP_Allocate (TCPA) Routine

The application calls TCP_Allocate routine to inform the TCP module how much data it is willing to accept on a given connection; that is, how much buffer space the application has available. The calling format shall be:

CALL TCPA (tcp_cepid, size, status , no_wait)

tcp_cepid This is the TCP CEPID which specifies this connection. The CEPID is an integer value whose range is -2**31 to 2**31-1.

size This is the number of bytes that the application will accept from the TCP.

status This is the returned status as a result of the allocation request issued by the application. The following integer values may be returned:

tcp_allocation_accepted
tcp_no_connection

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.2.6 TCP_Status (TCPS) Routine

The TCP_Status routine returns a copy of the status records that the TCP module is keeping for this connection. The calling format shall be:

CALL TCPS (tcp_cepid, user_cepid, tcp_sapid, user_sapid, source, * destination, ip_header, ip_options, ulp_timeout, tcp_stat_rec, status, no_wait)

tcp_cepid — This is the TCP CEPID which specifies this connection. The CEPID is an integer value whose range is -2**31 to 2**31-1.

user_cepid — This is the CEPID which specifies this connection. The CEPID is an integer value whose range is -2**31 to 2**31-1.

tcp_sapid — This is the SAPID returned by TCP in the open SAP request. The SAPID is an integer whose range is -2**31 to 2**31-1.

user_sapid — This is a record that contains the user SAPID. The SAPID is an integer whose range is -2**31 to 2**31-1.

source — This parameter specifies the internet address and port address of the datagram source. The internet address is optional, and in the multihost system, will specifiy the network solution used to send the datagram. There are a number of well known ports, an application not owning such a port should leave this field unspecified. All unspecified fields will be filled in and returned by TCP.

The internet address (ip_address) is a 32 bit value whose range is -2**31 to 2**31-1. The port address is a 16 bit value whose range is 0 to 2**15-1. There is a boolean value appended to the addresses that indicates whether a port address is specified. This value is set as follows: 0 = TRUE, 1 = FALSE.

destination

This parameter specifies the internet address and port address of the destination application.

The internet address (ip_address) is a 32 bit value whose range is -2**31 to 2**31-1. The port address is a 16 bit value whose range is 0 to 2**15-1. There is a boolean value appended to the addresses that indicates whether a port address is specified. This value is set as follows: 0 = TRUE, 1 = FALSE.

ip_header

This is a 19-word record that contains the IP header for the data gram. This record is used by the ULP to pass header information to IP. The fields to be completed by the ULP are specified below:

Word	Field Description	Range	ULP Supplied
0	version	0..15	
1	ihl	0..15	
2	precedence	0..7	yes
3	delay	Boolean	yes
4	throughput	Boolean	yes
5	reliability	Boolean	yes
6	unused_flag_1	Boolean	zero
7	unused_flag_2	Boolean	zero
8	total_length	0..0FFFF(16)	
9	identification	0..0FFFF(16)	yes

Word	Field Description	Range	ULP Supplied
10	unused_flag_3	Boolean	zero
11	dont_frag	Boolean	yes
12	more_frags	Boolean	
13	frag_offset	0..01FFF(16)	
14	time_to_live	0..255	yes
15	protocol	0..255	yes
16	checksum	0..FFFF(16)	
17	source	-80 000 000..7FFFFFFF(16)	
18	destination	-80 000 000..7FFFFFFF(16)	

ip_options — This is a 47-word record that contains the IP options for this connection. It consists of four subrecords for security, stream id, routing, and timing. The format of this record shall be as follows:

Word	Field Description	Range
0-5	ip_sec_rec	-
0	zero	0..127
1	in_use	Boolean
2	security	0..0FFFF(16)
3	compartments	0..0FFFF(16)
4	handling	0..0FFFF(16)
5	tcc	0..0FFFF(16)
6-8	ip_streamid_rec	-
6	zero	0..127
7	in_use	Boolean
8	streamid	0..0FFFF(16)
9-22	ip_routig_rec	-
9	zero	0.127
10	in_use	Boolean
11	rtype	0..255

Word	Field Description	Range
12	index	0..255
13	count	0..255
14-22	ip_list [array 0..8]	-80 000 000..7FFFFFFF(16)
23-46	ip_time stamp	-
23	zero	0..127
24	in_use	Boolean
25	ttype	0..255
26	index	0..255
27	count	0..255
28	overflow	0..255
29-46	sum_array [array 0..8]	ip_double_word
29-46	bg_array [array 0..3]	ip_full_stamp

ulp_timeout

tcp_stat_rec This parameter specifies a 13-word record containing the TCP status record. The record layout is as follows:

Word	Description	Range
0-1	ULP_timeout	ulp_timeout_rec
2	pdu_size	0-2**12
3	allocation	0-2**59
4	fsm_state	0-2**59
5	unsent	0-2**59
6	sent_unacked	0-2**59
7	sent_total	0-2**59

Word	Description	Range
8	undelivered	0-2**59
9	rec_unacked	0-2**59
10	rec_total	0-2**59
11	rndtrip_time	0-2**15-1
12	retrans_count	0-2**59

status — This is the returned status as a result of the status request issued by the application. The following integer value may be returned:

tcp_status_complete
tcp_no_connection

no_wait — Boolean value, if true, causes routine to return immediately after using request.

3.8.3.1.2.7 TCP_Disconnect (TCPD) Routine

The TCP_Disconnect routine is called by the application to terminate a connection. The application must wait until it receives a confirmation before assuming that the connection is dead. The application must also be willing to continue accepting data from the peer until the disconnect is confirmed. The calling format shall be:

```
CALL TCPD (tcp_cepid, status, no_wait)
```

tcp_cepid — This is the TCP CEPID which specifies this connection. The CEPID is an integer value whose range is -2**31 to 2**31-1.

status — This is the returned status as a result of the disconnect request issued by the application. The following integer value may be returned:

tcp_disconnecting
tcp_no_connection

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.2.8 TCP_Abort_Current_Connection (TCPACC) Routine

This TCP_Abort_Current_Connection routine is called by the application to abort a current connection. Unlike the disconnect routine, this routine disconnects immediately and any data in transit from the peer is thrown away. The calling format shall be:

```
CALL TCPACC (tcp_cepid, status, no_wait)
```

tcp_cepid — This is the TCP CEPID which specifies this connection. The CEPID is an integer value whose range is -2**31 to 2**31-1.

status — This is the returned status as a result of the abort request issued by the application. The following integer value may be returned:

tcp_aborted
tcp_no_connection

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.2.9 TCP_Send_Data (TCPSD) Routine

The TCP_Send_Data routine is called by the application to send data on an established connection. This routine has two special parameters, the PUSH flag and the URGENT flag. These two parameters tell TCP to give the data special processing. The calling format shall be:

```
CALL TCPSD (tcp_cepid, push, urgent, ulp_timeout, data, data_length,
            status, no_wait)
```

tcp_cepid — This is the TCP CEPID which specifies this connection. The CEPID is an integer value whose range is -2**31 to 2**31-1.

push — This flag is set TRUE to force the TCP module to immediately send the current data and any previous data to its peer. The flag is a boolean value, 0 = TRUE and 1 = FALSE.

urgent — When this flag is TRUE the data sent will be marked as special urgent data. The flag is a boolean value, 0 = TRUE and 1 = FALSE.

ulp_timeout

data — This is a pointer to the buffer that contains the data to be sent. It is assumed that the TCP module is given ownership of one copy of the buffer.

data_length — Integer count of number of words of data to send. Range is 0..2**32-1.

status — This is the status returned as a result of the send data request. The following integer value may be returned:

tcp_send_complete
tcp_no_connection
tcp_buffer_lost

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.2.10 TCP_Accept_Indication (TCPAI) Routine

The TCP_Accept_Indication routine is called by the application to check for indication arrival. The arriving indications are either the result of a peer connection request attempt (TCP_Connect_Indication) or a data indication from the peer (TCP_Data_Ind). The calling format shall be:

CALL TCPAI (id, buffer, length, status, no_wait)

id — This is the CEPID or SAPID for the TCP connection to poll. If -1, all TCP connections/SAPS are polled.

buffer — Buffer to be used to receive reply data.

length — Integer length of buffer supplied, in connection data units (default is 8-bit bytes).

STATUS — This is the status returned as a result of the acceptance of the indication. The following integer value may be returned:

tcp_indication_accepted
tcp_no_indication

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.2.10.1 TCP_Connection_Indication

The TCP_Connect_Indication is used by TCP to inform the application that a connection request has come in from another application on the network. The following information is supplied with the indication:

user_cepid — This is the Connection End Point Identifier (CEPID) which will be provided by TCP with all data passed to the application. CEPID is an integer value of range -2**31 to 2**31-1.

destination — This parameter specifies the internet address and port address of the destination application.

The internet address (ip_address) is a 32 bit value whose range is -2**31 to 2**31-1. The port address is a 16 bit value whose range is 0 to 2**15-1. There is a boolean value appended to the addresses that indicates whether a port address is specified. This value is set as follows: 0 = TRUE, 1 = FALSE.

ip_header — This is a 19-word record that contains the IP header for the data gram. This record is used by the ULP to pass header information to IP. The fields to be completed by the ULP are specified below:

Word	Field Description	Range	ULP Supplied
0	version	0..15	
1	ihl	0..15	
2	precedence	0..7	yes
3	delay	Boolean	yes
4	throughput	Boolean	yes
5	reliability	Boolean	yes

Word	Field Description	Range	ULP Supplied
6	unused_flag_1	Boolean	zero
7	unused_flag_2	Boolean	zero
8	total_length	0..0FFFF(16)	
9	identification	0..0FFFF(16)	yes
10	unused_flag_3	Boolean	zero
11	dont_frag	Boolean	yes
12	more_frags	Boolean	
13	frag_offset	0..01FFF(16)	
14	time_to_live	0..255	yes
15	protocol	0..255	yes
16	checksum	0..FFFF(16)	
17	source	-80 000 000..7FFFFFFF(16)	
18	destination	-80 000 000..7FFFFFFF(16)	

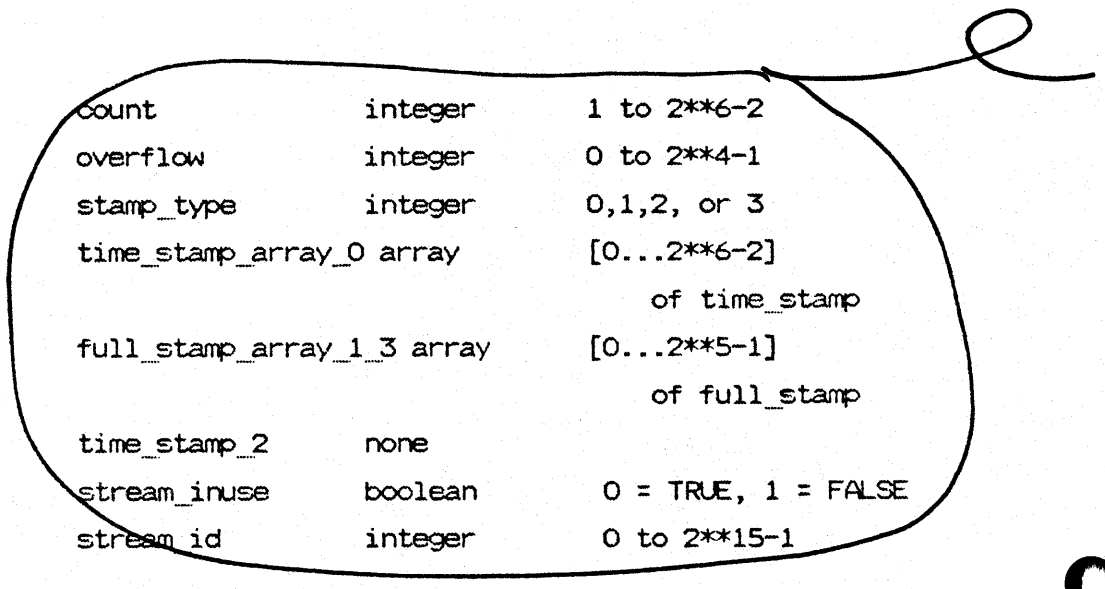
ip_options

This is a 47-word record that contains the IP options for this connection. It consists of four subrecords for security, stream id, routing, and timing. The format of this record shall be as follows:

Word	Field Description	Range
0-5	ip_sec_rec	-
0	zero	0..127
1	in_use	Boolean
2	security	0..0FFFF(16)
3	compartments	0..0FFFF(16)
4	handling	0..0FFFF(16)
5	tcc	0..0FFFF(16)
6-8	ip_streamid_rec	-
6	zero	0..127
7	in_use	Boolean
8	streamid	0..0FFFF(16)

Word	Field Description	Range
9-22	ip_routig_rec	-
9	zero	0.127
10	in_use	Boolean
11	rtype	0..255
12	index	0..255
13	count	0..255
14-22	ip_list [array 0..8]	-80 000 000..7FFFFFFF(16)
23-46	ip_time stamp	-
23	zero	0..127
24	in_use	Boolean
25	ttype	0..255
26	index	0..255
27	count	0..255
28	overflow	0..255
29-46	sum_array [array 0..8]	ip_double_word
29-46	bg_array [array 0..3]	ip_full_stamp

ulp_timeout



count	integer	1 to 2**6-2
overflow	integer	0 to 2**4-1
stamp_type	integer	0,1,2, or 3
time_stamp_array_0	array	[0...2**6-2] of time_stamp
full_stamp_array_1_3	array	[0...2**5-1] of full_stamp
time_stamp_2	none	
stream_inuse	boolean	0 = TRUE, 1 = FALSE
stream_id	integer	0 to 2**15-1

data	This is a pointer to the buffer containing the data sent with the connect request. If no data was sent then the pointer will be NIL. The application is assumed to accept ownership of one copy of this buffer.
data length	Integer value of length of data buffer in connection units.

3.8.3.1.2.10.2 TCP_Data_Ind

The TCP_Data_Ind is used by TCP to inform the application of data and non-connect request indications from an application on another network. The following information is supplied with the indication.

user_cepid	This is the user CEPID that was provided by the application at the time of the connect request. CEPID is an integer value of range -2**31 to 2**31-1.
ind_type	This is the type of indication that is being presented. The following types are defined as integer values of range 0 to 2**15-1.

tcp_disconnect_ind
tcp_disconnect_conf
tcp_data_ind
tcp_urgent_ind
tcp_error_ind
tcp_abort_ind
tcp_start_ind
tcp_stop_ind

data — This is a pointer to the buffer that contains the data. The ownership of one copy of the buffer is assumed to be accepted by the application. In the case of a data_indication or an urgent_indication, the buffer will contain the datagram received. In the case of an abort_indication or an error_indication, the buffer will contain an integer indicating the reason:

tcp_timeout
tcp_remote_abort

3.8.3.1.3 IP primitives

3.8.3.1.3.1 IP_Open_Sap (IPOS) Routine

The open SAP routine is used to register a Service Access Point (SAP) with the IP module. The IP module allows a total of 254 SAPs to be open at one time. Each SAP is identified by its SAPid value which is linked to the protocol value. The calling format shall be:

CALL IPOS (protocol, sapid, status, no_wait)

protocol — This value specifies the application protocol to the IP module. There are a number of well known values for authorized applications. Other applications should use the value zero, in which case the actual value will be assigned by the IP module, and will be returned. Protocol in an integer value whose range is 0 to 2**8-1.

sapid — This is a 32 bit value returned that identifies the particular IP SAP in all later requests. SAPID is an integer value of range -2**31 to 2**31-1.

status — This is the status of the open SAP request. The following integer values may be returned:

ip_sap_open
ip_protocol_inuse
ip_protocol_illegal

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.3.2 IP_Close_Sap (IPCS) Routine

The close SAP is used to terminate the use of an IP SAP. This frees up the SAP for use by another application. The calling format shall be:

CALL IPCS (protocol, sapid, status, no_wait)

protocol — This value specifies the application protocol to the IP module. There are a number of well known values for authorized applications. Other applications should use the value zero, in which case the actual value will be assigned by the IP module, and will be returned. Protocol in an integer value whose range is 0 to 2**8-1.

sapid — This is the SAPid returned on the original open SAP request. SAPID is an integer value whose range is -2**31 to 2**31-1.

status — This is the returned status of the close SAP request. The following integer values may be returned:

ip_sap_not_open
ip_sap_closed

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.3.3 IP_Send_Req (IPSR) Routine

The send data routine is used to send data out on the network through a a previously opened SAP. The address of this routine is returned to the application when the open request is made.

CALL IPSR (header, source, destination, options, sapid, data, data_length, status, no_wait)

header — This is the IP datagram header as defined in section 3.8.3.1.2.3.

source — This is the address of the sender. This address may be partially or completely unspecified. This address is specified when an application needs to indicate which network solution the IP should use if it is multihomed.

The source is a record made up of the following integer values:

field_inuse	integer	bit 0 = net bit 1 = host
network:	integer	0 to 2**20-1
host:	integer	0 to 2**27-1

destination — This is the address of the remote IP that the data is being sent to.

The destination is a record made up of the following integer values:

field_inuse	integer	bit 1 = net
		bit 2 = host
network:	integer	0 to 2**20-1
host:	integer	0 to 2**27-1

options — This is the IP options records as defined in section ~~3.8.1.2.3.~~ 3,8.3,1.2,3

sapid — This is the Service Access Point ID that was returned by the original open request. The SAPID is an integer value whose range is -2**31 to 2**31-1.

data — This is a pointer to the executive buffer returned containing the data portion of the message.

data_length — Integer length of data buffer in connection units.

status — This is the status of the request. The following integer values may be returned:

ip_net_unreachable
ip_host_unreachable
ip_fragmentation needed
ip_option_error
ip_sap_not_open
ip_source_illegal
ip_destination_illegal

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.3.4 IP_Accept_Indication (IPAI) Routine

The IP_Accept_Indication routine is called by the application to check for indication arrival. The arriving indications are either the result of a peer data indication (IP_Data_Ind) or an error indication (IP_Error_Ind).

CALL IPAI (sapid, no_wait)

sapid — This is the Service Access Point ID that was returned by the original open SAP request. SAPID is an integer value of range -2**31 to 2**31-1.

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.3.4.1 IP_Data_Ind

The IP_Data_Ind is used by the IP module to present data messages to the application. The following information is supplied with the indication:

header — This is the IP datagram header as defined in section 3.8.3.1.2.3.

source — This is the address of the IP that sent the data message.

The source is a record made up of the following integer values:

field_inuse	integer	bit 1 = net bit 2 = host
network:	integer	0 to 2**20-1
host:	integer	0 to 2**27-1

destination — This is the address of the receiver. This address indicates which network solution IP received the message from in a multi-homed system.

The destination is a record made up of the following integer values:

field_inuse	integer	bit 1 = net bit 2 = host
network:	integer	0 to 2**20-1
host:	integer	0 to 2**27-1

options — This is the IP options records as defined in section 3.8.3.1.2.3.

sapid — This is the SAPid that was returned by the original open request. SAPID is an integer value of range -2**31 to 2**31-1.

data — This is a returned pointer to the data buffer.

data_length — Length of data buffer in connection data units.

3.8.3.1.3.4.2 IP_Error_Ind

The IP_Error_Ind is used by the IP module to present error messages to the application. The following information is supplied with the indication:

(copy header, source, destination, options, sapid, data, data_length from previous ff)

count	integer	1 to 2**6-1
route_array	array	[0..2**6-2]
		of ip_address
time_stamp_inuse	boolean	0 = TRUE, 1 = FALSE
count	integer	1 to 2**6-2
overflow	integer	0 to 2**4-1
stamp_type	integer	0,1,2, or 3
time_stamp_array_0	array	[0..2**6-2]
		of time_stamp
full_stamp_array_1_3	array	[0..2**5-1]
		of full_stamp
time_stamp_2	none	
stream_inuse	boolean	0 = TRUE, 1 = FALSE
stream_id	integer	0 to 2**15-1

error — This is the type of error that occurred. The following errors are defined:

ip_net_unreachable
ip_host_unreachable
ip_protocol_unreachable
ip_port_unreachable
ip_fragmentation_needed
ip_route_failed
ip_timeout
ip_assembly_timeout
ip_option_error
ip_congestion

bad_param — This is the index of the bad parameter in the options list. The first parameter is precedence whose offset is 0. This is an integer value.

3.8.3.1.4 TELNET Primitives

3.8.3.1.4.1 TELNET Active Connect Request (TELACR)

```
telnet_sapid          (I)      : ITTSAPID,
user_cepid            (I)      : UCEPID,
source                (I/O)    : T$SADR$REC,
destination           (I)      : T$DADR$REC,
tcp_timeout_opt       (I)      : ULPTMOUT,
ip_header             (I)      : IP$HDR$REC,
ip_options            (I)      : IP$OPT$REC,
tcp_allocation        (I)      : ALLOC,
special_byte          (I)      : TNSPE$BYT,
perform_echo_opt      (I)      : TNECO$OPT,
log_connect_msg       (I)      : TNLOG$CON,
log_statistics        (I)      : TNLOG$STA,
peer_pcl_errs_alwd    (I)      : TNPER$ERR,
push                  (I)      : TPUSH,
urgent                (I)      : TPURG,
tcp_data              (I)      : ITTDATA,
data_length           (I)      : ITTSIZE,
telnet_cepid          (O)      : TCEPID,
status                (O)      : TNSTATUS,
no_wait               (I)      : boolean)
```

3.8.3.1.4.2 TELNET Abort Request (TELAR)

```
telnet_cepid          (I)      : TCEPID,
status                (O)      : TNSTATUS,
no_wait               (I)      : boolean)
```

3.8.3.1.4.3 TELNET Command Request (TELCR)

```
telnet_cepid          (I)       : TCEPID,
push                  (I)       : TPUSH,
tcp_timeout_opt       (I)       : ULPTMOUT,
command_code          (I)       : TNCOM$COD,
status                (O)       : TNSTATUS,
no_wait               (I)       : boolean)
```

3.8.3.1.4.4 TELNET Close Sap (TELCS)

```
telnet_sapid          (I)       : ITTSAPID,
status                (O)       : TNSTATUS,
no_wait               (I)       : boolean)
```

3.8.3.1.4.5 TELNET Disconnect Request (TELDR)

```
telnet_cepid          (I)       : TCEPID,
status                (O)       : TNSTATUS,
no_wait               (I)       : boolean)
```

3.8.3.1.4.6 TELNET Flow Control Request (TELFCR)

```
telnet_cepid          (I)       : TCEPID,
flow_cntl_status      (I)       : TFLO$CTL,
status                (O)       : TNSTATUS,
no_wait               (I)       : boolean)
```

3.8.3.1.4.7 TELNET Option Data Request (TELODR)

```
telnet_cepid          (I)       : TCEPID,
push                  (I)       : TPUSH,
tcp_timeout_opt       (I)       : ULPTMOUT,
option_code           (I)       : TNOPTION,
```

```
option_data          (I)      : TNOPT$DTA,
data_length          (I)      : ITTSIZE,
status               (O)      : TNSTATUS,
no_wait              (I)      : boolean)
```

3.8.3.1.4.8 TELNET Option Request (TELOR)

```
telnet_cepid         (I)      : TCEPID,
push                 (I)      : TPUSH,
tcp_timeout_opt      (I)      : ULPTMOUT,
option_negotiation   (I)      : TNOPT$NEG,
option_code          (I)      : TNOPTION,
status               (O)      : TNSTATUS,
no_wait              (I)      : boolean)
```

3.8.3.1.4.9 TELNET Open Sap (TELOS)

```
user_sapid           (I)      : USAPID,
telnet_sapid         (O)      : ITTSAPID,
status               (O)      : TNSTATUS,
no_wait              (I)      : boolean)
```

3.8.3.1.4.10 TELNET Passive Connect Request (TELPCR)

```
telnet_sapid         (I)      : ITTSAPID,
user_cepid           (I)      : UCEPID,
source               (I/O)    : T$SADR$REC,
destination          (I)      : T$DADR$REC,
tcp_timeout_opt      (I)      : ULPTMOUT,
ip_header            (I)      : IP$HDR$REC,
ip_options           (I)      : IP$OPT$REC,
tcp_allocation       (I)      : ALLOC,
special_byte         (I)      : TNSPE$BYT,
perform_echo_opt     (I)      : TNECO$OPT
```

```
log_connect_msg      (I)      : TNLOG$CON,
log_statistics       (I)      : TNLOG$STA,
peer_pcl_errs_alwd   (I)      : TNPER$ERR,
telnet_cepid         (O)      : TCEPID,
status               (O)      : TNSTATU,
no_wait              (I)      : boolean)
```

3.8.3.1.4.11 TELNET User Data Request (TELUDR)

```
telnet_cepid         (I)      : TCEPID,
push                 (I)      : TPUSH,
tcp_timeout_opt      (I)      : ULPTMOUT,
line_complete        (I)      : TNLIN$CMP,
process_text         (I)      : TNPRO$TXT,
user_data            (I)      : ITTDATA,
data_length          (I)      : ITTSIZE,
status               (O)      : TNSTATUS,
no_wait              (I)      : boolean)
```

3.8.3.1.4.12 TELNET Indication Primitive Calling Sequences

These routines are supplied by the application program.

3.8.3.1.4.12.1 TELNET Abort Indication (TELAI)

```
user_cepid           (I)      : UCEPID,
abort_code           (I)      : TNABORT
```

3.8.3.1.4.12.2 TELNET Command Indication (TELCI)

```
user_cepid           (I)      : UCEPID,
command_code         (I)      : TNCOM$COD
```

3.8.3.1.4.12.3 TELNET Connect Request Indication (TELCRI)

```
user_cepid          (I)     : UCEPID,
destination         (I)     : T$DADR$REC,
tcp_timeout_opt     (I)     : ULPTMOUT,
ip_header           (I)     : IP$HDR$REC,
ip_options          (I)     : IP$OPT$REC,
tcp_data            (I)     : ITTDATA,
data_length         (I)     : ITTSIZE)
```

3.8.3.1.4.12.4 TELNET Disconnect Confirm (TELDC)

```
user_cepid          (I)     : UCEPID)
```

3.8.3.1.4.12.5 TELNET Disconnect Indication (TELDI)

```
user_cepid          (I)     : UCEPID)
```

3.8.3.1.4.12.6 TELNET Error Indication (TELEI)

```
user_cepid          (I)     : UCEPID,
error_code          (I)     : TNERROR)
```

3.8.3.1.4.12.7 TELNET Flow Control Indication (TELFCI)

```
user_cepid          (I)     : UCEPID,
flow_cntl_status    (I)     : TFLO$CTL)
```

3.8.3.1.4.12.8 TELNET Option Data Indication (TELODI)

```
user_cepid          (I)     : UCEPID,
option_code         (I)     : TNOPTION,
option_data         (I)     : TNOPT$DTA,
data_length         (I)     : ITTSIZE)
```

3.8.3.12.1.4.9 TELNET Option Indication (TELOI)

```
user_cepid          (I)     : UCEPID,
option_negotiation  (I)     : TNOPT$NEG,
option_code         (I)     : TNOPTION)
```

3.8.3.1.4.12.10 TELNET User Data Indication (TELUDI)

```
user_cepid          (I)     : UCEPID,
line_complete       (I)     : TNLIN$CMP,
user_data           (I)     : ITTDATA,
data_length         (I)     : ITTSIZE)
```

3.8.3.1.5 Common Routines

Several common routines have been identified for use by DDN system applications such as FTP, SMTP, and Supervisor. These routines are not supported for users nor do they appear in the User Manual.

3.8.3.1.5.1 DDN Abort Processing - DDNABT

DDNABT is used to abort a DDN application due to some irrecoverable error. First, NETDMB is called to dump the program. Next, the message is issued to the dayfile, and if possible, to the DDN Supervisor. Then, any NET files (trace,statistics) are flushed and an abort is issued.

```
CALL DDNABT (message)
```

message — A character string, up to 40 characters long and zero-byte terminated, containing the reason for the abort. Every call to DDNABT must use a unique message (within the program).

3.8.3.1.5.2 Check QTGET response - DDNCQTG.

DDNCQTG is used to perform standard processing following a QTGET call. It performs the following tasks:

a. Handles DDN specific data. This includes asnynchronous or synchronous supervisory messages for connections that are maintained by the DDN runtime routines DDNxxxx, TCPxxxx, IPxxxx, TELxxxx, SMTPxxx, and FTPxxxx.

b. Logs (using DDNLM) certain error status from QTGET, including QTGET errors 6,13,23,63, all 'reserved' numbers, and all numbers that are not returned by QTGET (5,11,12,15,22,43,44).

c. Processes debugging and statistics requests from NAM.
If a 36 is received, QTDMB is called.
If a 37 is received, DEBUG is set true in DDNCOM.
If a 38 is received, DEBUG is set false in DDNCOM.
If a 39 is received, QTSTC is called.
If a 40 is received, QTREL is called.

d. Maintains the connection state in the NCT entry.
If a 6 is received, sets 'unused' (terminated) state.
If a 20 is received, sets 'unused' (terminated) state.
If a 42 is received, sets 'inactive' state.

e. Maintains the network state in DDNCOM.
If a 9 is received, sets 'idledown' state.
If a 10 is received, set 'terminated' state.

f. Continues processing initiated by a DDNLINK request. The following error codes are processed if the connection is started by a DDNLINK call, and otherwise ignored:

If a 0 is received and the connection state is 'connected' then the data is validated as an application ID. The connection state is either advanced to 'identified' or the connection is terminated. If 'identified' and no ID has previously been sent, it is sent to the app now.

If a 2 or 14 is received, the connection state is advanced to 'connected', and an Application ID is sent if appropriate.

g. The following are not processed (unless DDN-runtime managed): 0,1,2,7,8,14,18,19,21,24,25,26,27,28,31,32,33,34,35.

h. Even though DDNCQTG does some processing on certain codes, the application may also have to do processing on the codes. If the application should not do processing on the QTGET data, DDNCQTG will reset the return code to 1. Following are recommended actions to be taken by an application for codes that are partially processed by DDNCQTG:

2 - The application should send (or wait for) data depending on the protocol. DDNCQTG will not return a '2' status until application IDs have been exchanged.

6 - Additional clean-up by the application will probably be required after a broken connection, particularly if there are related connections open that must be informed or closed.

9 - The application should send termination notices to all applicable connections and abort any passive TCP/IP/TELNET or otherwise idle connections.

10 - Files should be flushed and the program should terminate.

13 - Send necessary error responses to users and clean up tables.

14 - The application should send (or wait for) data depending on the protocol. DDNCQTG will not return a '14' status until application IDs have been exchanged.

39 - The application should reset its own statistical counters.

42 - The application, if appropriate, should poll the other user to determine if still present, or possibly close the connection (resonse to this case is TBD for FTP/SMTP).

The calling sequence for DDNCQTG is:

CALL DDNCQTG (buffer, status)

buffer — buffer supplied in original QTGET call.

status — Return status. This is the status in the NIT from the QTGET call and possibly changed by DDNCQTG.

3.8.3.1.5.3 Get DDN Network Data - DDNGET

DDNGET is used to receive data from applications, or to receive asynchronous supervisory messages. DDNGET issues a QTGET with the supplied parameters and then calls DDNCQTG to process the return status.

CALL DDNGET (buffer, length, acn, status, no_wait)

buffer — Array to receive data.

length — Length of buffer in connection data units (default is 8-bit bytes). In general, this should be the maximum NAM block size.

acn	Connection number to be polled. If zero, only asynchronous messages are polled. If -1, data is accepted from any connection, and acn is updated by DDNGET to the acn for data received.
status	Return status from the QTGET/DDNCQTG call.
no_wait	Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.5.4 Application Idle Processing - DDNIDLE.

DDNIDLE is used to handle rollout and recall processing for DDN applications. RECALL is called for a period of time, after which if no network data has arrived, then NETWAIT is called to rollout the application. The rollout and recall parameters are set with defaults in common block DDNCOM and can be changed dynamically by the application if desired. DDNIDLE takes no parameters.

```
CALL DDNIDLE
```

3.8.3.1.5.5 Application Connection Processing - DDNLINK

DDNLINK is used to connect to another DDN application, and to establish which protocols are supported for incoming connection requests. Only one DDNLINK or QTLINK can be outstanding at one time. DDNLINK performs a QTLINK automatically and additionally, through DDNCQTG, sends the application ID and verifies the ID sent by the connecting application. When an ID has been sent and a valid ID has been received, DDNCQTG returns a 2 or 14 code (depending on which application initiated the connection) indicating that the connection is ready for data transfer.

CALL DDNLINK (application_name, destination_host, outcall, length, protocol, status, no_wait)

application name — 0-7 characters containing name of application with which to connect. If zero, then this call specifies or de-specifies a protocol for which the application will accept inbound connection requests. Application_name is the NAME1 portion of NAM connections.

destination_host — 0-3 characters containing destination host. This should be zero for intra-host connections. Destination_host is the NAME2 portion of NAM connections.

outcall — Buffer containing outcall parameters. Only SSJ= jobs can send outcall parameters. For intra-host connections, the first byte of user call data is reserved for SSJ= identification. Outcall parameters other than user call data should not be used for intra-host connections.

callen — Length of outcall buffer in 8-bit bytes. Zero if no outcall parameters are specified.

protocol — Protocol identifier and version. See NCT for description. For incoming connection requests, a call must be made to DDNLINK (with appname=0) for every protocol that the application supports. These calls must be made after the DDNSSA call but before the DDNRA call. If an incoming appl-id does not match protocol id or version, then the mismatch is logged and the connection is broken.

status	Status from QTLINK.
no_wait	Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.1.5.6 Send DDN Network Data - DDNPUT.

DDNPUT is used to send data or supervisory messages to an application. DDNPUT calls QTPUT with the supplied parameters.

CALL DDNPUT (buffer, length, acn, endmsg, qual, status, no_wait)

buffer	Array buffer containing data or messages to send.
length	Number of connection data units in buffer (default is 8-bit bytes).
acn	Connection number over which to send data.
endmsg	End-of-message flag. Boolean value, if true, this is the last (or only) block in a message.
qual	Qualified data flag. Boolean value, if true, the data is sent as qualified data.
no_wait	Boolean value. Unused, kept for consistency.

3.8.3.1.5.7 Set System Application - DDNSSA.

DDNSSA allows an application to establish itself as a DDN system application. This means that the application can use the standard DDN interface routines (for example, TCPxxxx) and can also use the QTRM routines and DDNGET/DDNPUT routines. DDN applications that use both types of routines are responsible for allocating the NIT and NCT. They are also responsible

for using DDNGET instead of QTGET so that data destined for connections handled by the interface routines (for example, TCPxxxx) can be processed. The application should not do anything directly with these connections (they are identified by a flag in the NCT).

CALL DDNSSA (nit, nitlen, nct, nctlen, sysflag, no_wait)

nit — Array to be used for the Network Information Table. The application should not set any values into this table; it will be initialized by DDNSSA/DDNRA.

nitlen — Number of 60-bit words in the NIT. Should be (n+1)*10 where n is the total number of NAM connections the application will support. Note that one is required for each anticipated TCP, IP, and TELNET connection, plus 1 connection if the SMTPxxx interface is used, plus 1 connection if the FTPxxxx interface is used, plus 1 connection for DDN registration, plus the number of application-specific connections to be supported.

nct — Array to be used for the Network Control Table.

nctlen — Number of 60-bit words in the NCT. Should be n*TBS where n is the same as for nitlen. TBS must be at least ??? words long, but can be longer if application-specific entries have been defined.

sysflag — System Application flag. Should be true if the application does QTxxxx and/or DDNPUT/DDNGET calls in addition to DDN Interface calls (for example, TCPxxxx, FTPxxxx). Examples of this are FTPI,FTPS.

no_wait — Boolean value. Unused, kept for consistency.

3.8.3.1.5.8 End Single Connection - DDN ENDT

CALL DDNENDT (acn, no_wait)

acn — Connection number to be closed.

no_wait — Boolean value, if true, causes routine to return immediately after issuing request.

3.8.3.2 DDN Host Name Table

The Gateway name server function shall access the DDN host name table. This table is maintained as a file, DDNNAME under un=LIBRARY, with public read access. The file is initially created as part of the Gateway installation process with a single entry. This entry identifies the Network Information Center host where the real host name table is maintained, as defined in RFC-810. Whenever the DDN supervisor is initiated, it checks to determine if the file is the installation file or a full name table file. If only an installation file, the Supervisor will automatically initate an FTP request to acquire the file. The network administrator can obtain updated copies by terminating the Supervisor, purging the file, and re-starting the Supervisor.

3.8.3.3 Network Access Method

The Gateway shall use Network Access Method (NAM) Application Interface Program (AIP) interfaces to establish connections and exchange data between AI, Supervisor, and DDN/C170 Gateway.

3.8.3.4 Network Definition File

G/W shall define its NAM application parameters via NDL configuration directives. The directives used are defined in subsection 3.8.8.

3.8.3.5 Operator Interface

FTP shall allow the operator to exercise some control over which messages are written to the log file. The following are the K-display HOP commands used to control message wiring to the log file.

K.DE = FTP	Write error messages only to the log file
K.DB = FTP	Write debug and error messages to the log file
K.RS = FTP	Write statistics, debug, and error messages to the log file

Error messages are always written to the log file regardless of any operator intervention.

If the application has been loaded using NETIOD, the K.RS command will also restart AIP statistical gathering.

3.8.4 **Internal Interfaces**

The following subsections describe the interfaces between the major components of DDN Gateway. The internal interfaces identified are:

Application Interface to DDN/C170 Gateway
Application Interface to DDN Supervisor

3.8.4.1 Application Interface to DDN/C170 Gateway

The application interface to DDN/C170 gateway interface consists of packets of data passed via a NAM connection. Data packets sent from AI to G/W contain data provided by the application in TCP/IP/TELNET primitives along with a primitive identifier and are destined for TCP/IP/TELNET in the MDI. Data packets sent from G/W to AI contain data provided by TCP/IP/TELNET indications along with an indication identifier and are destined to an application program.

A NAM connection exists between the Gateway and each AI in an application that has registered for host protocol services. An additional connection exists for each TCP, IP, or TELNET connection, whether established or pending.

The following interface conventions are defined for data passed between the MDI-resident Gateway and the C170 Interface routines. In the discussion, a byte always refers to a CDCNET 8-bit byte; a word always refers to a CYBER 60-bit word.

a. A message sent between MDI and C170, in either direction, will consist of a header part and an optional data part. The header part will contain information about the message length, primitive/indication type, and most of the parameters to or from the primitive/indication. The data part will be actual data. Since most primitives/indications do not include data buffers, most indication/primitive messages do not contain data parts.

b. The header part will always be a multiple of CYBER words. This implies that the header will always be a multiple 15 bytes and a multiple of 2 words.

c. The data part will always be a multiple of CYBER words also, just like the header. If the actual amount of data is less than a multiple of 15 bytes, the message will be padded with zero bytes. The actual data length will be stored in a field in the header. There is no requirement on the number of bytes of actual data. If the header plus data size exceeds the maximum NAM block size then a multi-block message will be used to send the data. Whether BIP does this automatically or whether gateway will have to do the blocking, in an MDI-to-C170 transfer, is unknown at this time. In a C170-to-MDI transfer, NAM does not do blocking and so it must be done by the interface routines. No limit is to be placed on the maximum data buffer size (other than actual machine resource or operating system limits).

d. No field within a header part will cross a CYBER word boundary. That is, every field must be completely contained in a 60-bit block (7 1/2 bytes). The position of a header field within a CYBER word, and its size, can be chosen according to the convenience of the gateway and indication/primitive data structure requirements, since SYMPL can be used to arbitrarily define field position/size. Unused filler bits must be explicitly defined in the CYBIL definition of the interface; they should not be defined in the SYMPL definition.

e. Messages from C170-to-MDI are either primitive-requests or indication-responses. Messages from MDI-to-C170 are either indication-requests or primitive-responses. For each request received, exactly one response must be returned. No more than one unacknowledged (that is, no response received) request can be outstanding at each end of the interface. Note that the MDI and C170 issue requests asynchronously, so the C170 can receive an indication-request from the MDI while waiting for a primitive-response, and the MDI can receive a primitive-request while waiting for an indication-response.

f. The header field always contains the following minimum information:

1) Primitive/Indication identifier (1-7 characters which match the C170 interface routine name).

2) Block number. This is a 16-bit modulo integer that is incremented by the sender each time a request is sent. The receiver will always check this number against the previous number received to determine if blocks have been lost. The initial 'previous' number (that is, when no block has yet been received) is zero. If gateway receives an out-of-order block, it will ignore it and return an error response. If the C170 receives an out-of-order block, or if it receives an error response for an out-of-order block, it will reset the connection and return status to the user.

3) Return status (always zero for request, non-zero for response if an error).

4) Number of header bytes. This is required due to some variable-length fields in the header (that is, IP Timestamp). If a field is variable, there must be a corresponding field that specifies its length.

5) Number of actual data bytes in data part (zero if no data).

6) Primitive/Indication parameters (except data buffer), where output parameters are not defined in the request but are returned in the response along with the original input parameters. Therefore, the format of the header is identical for a request and its corresponding response.

g. When the C170 first establishes a connection with MDI, a standard application identification package will be supplied that identifies the application name, protocol type, and version number. Gateway will verify this connection info. This packet will be defined in another ERSIU.

3.8.4.2 Application Interface to DDN Supervisor

The application interface to DDN Supervisor interface consists of requests and responses passed via NAM connections, with the requests sent by AI and responses returned by the Supervisor. The requests are for registration, name translation, and message logging with associated data supplied by the application. The responses from Supervisor contain requested information and reply codes that correspond to the request.

A NAM connection exists between each instance of a AI in a registered application and the Supervisor.

3.8.5 Errors and Error Recovery

3.8.5.1 Supervisor Failure and Recovery

The DDN Supervisor shall self-start upon failure. A dump shall be taken of the failed job prior to re-start and the failure time shall be logged. After two consecutive failures within 10 minutes, the re-start procedure shall issue a NAM operator alert and refuse all subsequent calls. The problem must then be corrected and the Supervisor manually re-started. Applications registered at the time of the failure shall be notified via the 'not_registered' reply; subsequent action is application dependent.

3.8.5.2 DDN/C170 Failure and Recovery

The Gateway shall follow standard CDCNET conventions for process failure. Applications using the TCP/IP/TELNET services at the time of the failure shall be notified via the 'network_unavailable' reply; subsequent action is application dependent.

3.8.6 Data Structures

Gateway data structures are TBS.

3.8.7 Equipment Configuration

The Gateway CPCI has no special equipment configuration requirements.

3.8.8 Installation

The following subsections define the installation procedures and parameters for Gateway.

3.8.8.1 G/W Installation Procedure

G/W is installed using the DDNGW build procedure and build verification is performed using the VDGW procedure. DDNGW should be treated as a Group 3 job as defined in the NOS Installation Handbook. The G/W product PL is maintained in MODIFY format. The following files are created or updated by the build procedure: JOBGW, DDNNAME, DDNLIB(updated).

3.8.8.2 G/W Installation Parameters

The following installation parameters can be tailored for specific sites. They are located in common deck COMVIPR.

IPGCTO — Gateway Connection Timeout. This is the maximum time in seconds that G/W will wait for a remote host protocol to respond to a connect request. After this period, the connection will be aborted. The default is 600.

IPMBUF — Maximum Buffer Size. This is the buffer size used for hot protocol transfers. The minimum size is the NAM downline block size (255 words default). The default value is 1024B.

LOGMSG — Message logging category. This defines the level of messages normally logged, barring operator intervention. The possible values are:

0 = log error messages only (the default)
1 = level 0 + debug messages
2 = level 1 + statistical messages

The following modify symbols can be specified using the *DEFINE directive to influence the G/W build process:

DEBUG — If defined, G/W will generate AIP trace messages.

IMS — If defined, the G/W listing will contain Internal Maintainance information.

STATS — If defined, G/W will generate AIP statistics messages.

3.8.8.3 Network Startup Master File

The following directives must be placed in the Network Startup Master File if automatic initiation of G/W is required.

```
JOB(JOBDDNS,DDNS)
```

3.8.8.4 Network Definition File Directives

The following table lists the Gateway modules and their configuration parameters:

Name	Description	Number of Copies (MXCOPYS)	Request Startable	Privileged
DDNS	DDN Supervisor	1	yes	yes
DDNA	DDN User Application	any(1)	no	no
DDNG	DDN Gateway (this will eventually reside in CDCNET and will require X.25-style INCALL/OUTCALL directives)	1	no	yes

Additionally, Gateway application users must provide OUTCALL directives to G/W for their applications.

3.8.8.5 Gateway Transmittals

3.8.8.5.1 Gateway/C170 Transmittal

FTP/SMTP/CDCNET GW transmittals all depend upon the installation of this transmittal.

a. DDN Gateway C170 Program Library (DGWnnnn). This is an UPDATE formatted program library containing the Gateway C170 interface routines, DDN Supervisor, and common code used by more than one DDN component. The format of this program library is described below. The following object is generated from this PL:

Name	Type	Location	Description
DDNS	Program	System	DDN Supervisor
DDNTEXT	SYMPL Text	None	DDN Common SYMPL Declarations
DDNLIB	Library	System	DDN Application Interface Library
NAMSTRT	Procedures	un=NETOPS	NAM Startup Proc. Update for DDNS

b. DDN C170 Gateway Installation procedure (DDNGW). This procedure follows DECKOPL conventions to install the DDN C170 Gateway.

c. DDN C170 Gateway Installation Verification procedure (VDDNGW). This procedure is to be executed on a newly-built system to verify that DDN C170 Gateway has been properly installed. It does not verify that the Gateway actually works. It follows DECKOPL verification conventions.

3.8.8.5.2 Gateway/CDCNET Transmittal

No other transmittal can depend upon this transmittal.

a. DDN Gateway CDCNET Program Library (DGNnnnn). This is an SES MODIFY formatted PL containing the Gateway CDCNET-resident program and command processors. The decks in this PL are incorporated into the CDCNET PL by a build procedure; it is not used stand-alone. The following object is generated from this PL:

Name	Type	Location	Description
		(TBS)	

b. CDCNET GW Installation procedure (DDNGN). This procedure follows TBS CDCNET build procedure conventions to install the Gateway in the CDCNET PL and system.

c. CDCNET GW Installation Verification procedure (VDDNGN). This procedure is to be executed on a newly-built system to verify that the Gateway has been properly installed. It does not verify that the gateway actually works. It follows TBS CDCNET verification conventions.

3.8.8.6 Program Library Structures

3.8.8.6.1 PL Structure

a. Where deck lists are described as 'organized', it means that the decks are arranged in a SIMPLE manner logical with the structure of the program. For example, the main program followed by all primary routines in alphabetical order followed by all minor routines in alphabetical order. If no structure can be identified, 'organized' degenerates to 'alphabetized', never to 'random' or 'complex'.

b. DDN Gateway/C170 PL Structure

```
INFO
HISTORY
[alphabetized common decks]
-TEXTS-     [contains *CWEOR]
DDNTEXT
-LIBSYMP-   [contains *CWEOR]
[SYMPL library routines, alphabetized]
-LIBCOMP-   [contains *CWEOR]
[COMPASS library routines, alphabetized]
-SYMPL-     [contains *CWEOR]
-DDNSSYMP-  [empty deck]
[DDN Supervisor SYMPL routines, organized]
-COMPASS-   [contains *CWEOR]
-DDNSCOMP-  [empty deck]
[DDN Supervisor COMPASS routines, alphabetized]
-NAMSTRT-   [contains *CWEOR]
JOBDDNS
-ENDPL-     [empty deck]
```

NOTE: All non-library SYMPL/COMPASS decks are compiled into a large library. The DDNS main program is loaded using LIBLOAD from that library.

c. Inter-Component Common Decks

DDN System-Wide

COMVDDN	DDN General Runtime Interface Definitions
COMVDIP	DDN System Installation Parameters
COMVFET	CIO FET Definitions
COMVFIT	CRM FIT Definitions
COMVIPC	IP Interface Definitions

COMVNCT	DDN NCT Definitions
COMVSYS	CYBER System Definitions
COMVTCP	TCP Interface Definitions
COMVTEL	TELNET Interface Definitions

DDN Gateway

COMVASC	DDN Application to Supervisor Interface
COMVGIP	DDN Gateway Installation Parameters
COMVMDI	DDN MDI to C170 Interface

3.8.8.6.2 PL Structure

a. Where deck lists are described as 'organized', it means that the decks are arranged in a SIMPLE manner logical with the structure of the program. For example, the main program followed by all primary routines in alphabetical order followed by all minor routines in alphabetical order. If no structure can be identified, 'organized' degenerates to 'alphabetized', never to 'random' or 'complex'.

b. Gateway/CDCNET PL Structure.
(TBS)

c. Inter-component Common Decks.
(TBS)

3.9 Network Operating System (NOS)

The following list shows the applications which initiate 3.9.??.?? ???????????

The following list shows the applications which initiate connections, and the number of connections that may be initiated from each instance of an

application. This is used as the basis for the development of INCALL/OUTCALL directives and test configurations.

FTP -> FTPI (1)	SMTP -> SMPI (1)
FTP -> DDNS (1)	SMTP -> DDNS (1)
FTPA -> FTPI (1)	SMPA -> SMPI (1)
FTPA -> DDNS (1)	SMPA -> DDNS (1)
FTPI -> DDNS (1)	SMPI -> DDNS (1)
FTPI -> DDNG (n)	SMPI -> DDNG (n)
FTPI -> FTPS (1)	SMPS -> SMPI (1)
FTPS -> DDNG (1)	DDNA -> DDNG (n)
	DDNA -> DDNS (1)

NOTE: The number following the connection is the number of connections per instance of the application PAIR. For example, there can be many instances of an SMTP, but there is only one connection per instance. The number of instances of each application has been identified in the previous table. The application before the arrow is the application that initiates a connection.

3.9.1 NOS Abstract

NOS version 2.4.2 will be the operating system for DDN. A pre-release is expected in May 85 with a formal release in Aug of 85.

3.9.2 NOS Modifications Installation

3.9.2.1 NOS Modifications Transmittal

FTP/SMTP/C170 GW/CDCNET GW transmittals all depend upon the installation of this transmittal.

a. File containing modsets to NOS OPL. Changed decks are currently MODVAL and COMTNAP. At a minimum, MODVAL and NVF (a NAM module) must be re-built for these modsets to take effect. The modset file is in standard MODIFY multi-record format, according to standard. These modsets can be installed using the standard DECKOPL NOS installation procedure.

b. NOS Mods Installation Verification procedure (VDDNNOS). This procedure is to be executed on a newly-built system to verify that the NOS mods have been properly installed. It does not verify that the modsets actually work. It follows DECKOPL verification conventions.

c. File containing modsets to NAM. Changed decks are currently QTRMNIT, QTGET and QTOPEN. At a minimum, these decks must be re-built for the modsets to take effect (note that QTRMNIT must be assembled prior to assembly of the others). The object code from these decks resides on libraries NETIO and NETIOD. The file is in UPDATE SLASHPL format, according to standard. These modsets can be installed using the standard DECKOPL NAM/NAMD installation procedure.

d. NAM Mods Installation Verification procedure (VDDNNAM). This procedure is to be executed on a newly-built system to verify that the NAM mods have been properly installed. It does not verify that the modsets actually work. It follows DECKOPL verification conventions.

3.9.2.2 PL Structure

a. Where deck lists are described as 'organized', it means that the decks are arranged in a SIMPLE manner logical with the structure of the program. For example, the main program followed by all primary routines in alphabetical order followed by all minor routines in

alphabetical order. If no structure can be identified, 'organized' degenerates to 'alphabetized', never to 'random' or 'complex'.

b. NOS Modifications PL Structure.

(TBS)

c. Inter-Component Common Decks

DDN System-Wide (defined in DDNTEXT)

COMVDDN	DDN General Runtime Interface Definitions
COMVDIP	DDN System Installation Parameters
COMVFET	CIO FET Definitions
COMVFIT	CRM FIT Definitions
COMVIPC	IP Interface Definitions
COMVNCT	DDN NCT Definitions
COMVSYS	CYBER System Definitions
COMVTCP	TCP Interface Definitions
COMVTEL	TELNET Interface Definitions

DDN Gateway

COMVASC	DDN Application to Supervisor Interface
COMVGIP	DDN Gateway Installation Parameters
COMVMDI	DDN MDI to C170 Interface

FTP

COMVAPC	FTP Application to Protocol Interpreter Interface
COMVFTP	FTP Definitions
COMVIPF	FTP Installation Parameters
COMVPFC	FTP Protocol Interpreter to File Server Interface
COMVSSJ	FTP NOS Validation Interface

COMVAPC	SMTP Application to Protocol Interpreter Interface
COMVIPM	SMTP Installation Parameters
COMVSMP	SMTP Definitions

3.10 Distributed Communications Network Software (DCNS)

3.10.1 DCNS Abstract

DCNS is CDC's standard product communication software that will reside in the Mainframe Device Interface (MDI), Terminal Device Interface (TDI), and Network Device Interface (NDI) hardware. For DDN, the NDI will be supplemented with X.25, TELNET, TCP, and IP software in compliance with requirements.

3.10.2 CONVERT Modifications Transmittal

CDCNET GW transmittal depends upon the installation of this transmittal.

a. File containing modsets to CDCNET PL. Changed decks are currently TBS (incl. TCP, IP, TELNET, BIP/SVM). The modset file is in standard SES MODIFY multi-record format, according to standard. Installation of these modsets is TBS after CDCNET build conventions are known.

b. CDCNET Mods Installation procedure (DDNCON). This procedure follows TBS CDCNET build procedure conventions to install the modsets in the CDCNET PL and system.

c. CDCNET Mods Installation Verification procedure (VDDNCON). This procedure is to be executed on a newly-built system to verify that the CONVERT mods have been properly installed. It does not verify that the modsets actually work. It follows TBS CDCNET verification conventions.

3.10.3. PL Structure

a. Where deck lists are described as 'organized', it means that the decks are arranged in a SIMPLE manner logical with the structure of the program. For example, the main program followed by all primary routines in alphabetical order followed by all minor routines in alphabetical order. If no structure can be identified, 'organized' degenerates to 'alphabetized', never to 'random' or 'complex'.

b. CONVERT PL Structure.

(TBS)

c. Inter-component Common Decks.

(TBS)

3.11 CDCNET Hardware

3.11.1 CDCNET Hardware Abstract

The CDCNET hardware consists of a device termed a Device Interface (DI). The DI is a modular and configurable microprocessor. Depending on the configuration a DI may be a:

MDI - Mainframe Device Interface, which connects the DI to a CYBER.

NDI - Network Device Interface, which acts as a packet switching device to transfer data between network elements.

TDI - Terminal Device Interface, which is used to interface terminals to the network.

The following minimum DCN hardware is required for a DDN CYBER Host Interface:

Qty.	Part No.	Name	DI Variant	Description
3	2601-003	DI	MDI,NDI,TDI	Device interface: includes master processor board, power supply, cabinet and internal bus system.
1	2603-001	SMM	NDI	512K byte main memory module
2	2604-001	SMM	MDI,TDI	1024K byte main memory module
1	2606-001	PMM	MDI	Private memory module
1	2607-001	MCI	MDI	Mainframe channel interface
3	2608-002	ESCI	MDI,NDI,TDI	Ethernet system communication interface
2	2609-001	CIM	NDI, TDI	Communication interface module
1	2610-016	LIM	NDI	RS-449 line interface module
1	2612-017	LIM	TDI	RS-232 line interface module