3.3.8.2 Called Facility Designation parameter

3.3.8.2.1 This parameter contains the addressed ground system's facility designation.

3.3.8.2.2 The *Called Facility Designation* parameter value shall conform to the abstract syntax four- to eight-character facility designation.

3.3.8.3 Calling Facility Designation parameter

3.3.8.3.1 This parameter contains the sending ground system's facility designation.

3.3.8.3.2 The *Calling Facility Designation* parameter value shall conform to the abstract syntax four- to eight-character facility designation.

3.3.8.4 ASE Version Number parameter

3.3.8.4.1 This parameter contains the version number of the CPDLC-ASE.

3.3.8.4.2 When provided by the CPDLC-ground-ASE, the *ASE Version Number* parameter shall conform to the abstract integer value in the range of 1-255.

3.3.8.4.3 Only if the sending CPDLC-ground-ASE version number is not equal to the receiving CPDLC-ground-ASE version number shall the receiving CPDLC-ground-ASE version number be confirmed to the sending CPDLC-ground-user.

Note.— If the sending CPDLC-ground-ASE version number is the same as the receiving CPDLC-ground-ASE version number, the Version Number parameter is not present in the indication given to the receiving CPDLC-ground-user or in the confirmation given to the sending CPDLC-ground-user.

3.3.8.5 CPDLC Message parameter

3.3.8.5.1 The sending CPDLC-ground-user uses this parameter to forward a CPDLC message to another CPDLC-ground-user.

3.3.8.5.2 The *CPDLC Message* parameter value shall conform to the ASN.1 abstract syntax ATCForwardMessage.

3.3.8.6 Class of Communication Service parameter

3.3.8.6.1 This parameter contains the value of the required class of communication service. If not specified by the CPDLC-ground-user, this indicates that there is no routing preference.

3.3.8.6.2 Where specified by the CPDLC-ground-user, the *Class of Communication Service* parameter shall have one of the following abstract values: "A", "B", "C", "D", "E", "F", "G" or "H".

3.3.8.7 Security Required parameter

3.3.8.7.1 The *Security Required* parameter contains the value of the required level of security, if specified by the CPDLC-ground-user.

3.3.8.7.2 If the received *Security Required* parameter is not as expected per the local security policy, the receiving CPDLC-ground-ASE will abort.

3.3.8.7.3 Where specified by the CPDLC-ground-user, the *Security Required* parameter shall have one of the following abstract values: "no security" or "secured exchange".

Note.— Where not specified by the CPDLC-ground-user, this indicates that there is no security required.

3.3.8.8 Result parameter

3.3.8.8.1 The *Result* parameter contains the result of the CPDLC-forward service. It will indicate success (service supported and matching versions), service unsupported, or version number incompatibility.

3.3.8.8.2 The *Result* parameter value shall conform to the ASN.1 abstract syntax ATCForwardResponse.

3.3.9 CPDLC-user-abort service

3.3.9.1 General

3.3.9.1.1 This service provides the capability for either the CPDLC-air-user or the CPDLC-ground-user to abort communication with its peer. It can be invoked at any time the CPDLC-user is aware that the CPDLC service is in operation. The CPDLC-user-abort service can be used for operational or technical reasons. It is an unconfirmed service. Messages in transit may be lost during this operation.

3.3.9.1.2 If the service is invoked prior to complete establishment of the dialogue, the CPDLC-user-abort indication may not be provided. A CPDLC-provider-abort indication may result instead.

3.3.9.1.3 The CPDLC-user-abort service shall contain the primitives and parameters as presented in Table 3-8.

Table 3-8. CPDLC-user-abort service parameters

Parameter name	*Req*	*Ind*
Reason	U	M

3.3.9.2 Reason parameter

3.3.9.2.1 The *Reason* parameter is used to indicate a reason for aborting the CPDLC or DSC dialogue.

3.3.9.2.2 If provided by the CPDLC-user, the parameter indicated to the peer CPDLC-user is that provided by the CPDLC-user; otherwise, it is what the ASE supplies.

3.3.9.2.3 The *Reason* parameter value shall conform to the ASN.1 abstract syntax CPDLCUserAbortReason.

3.3.9.2.4 When this parameter is provided by the CPDLC-user, the same value shall be indicated to the peer CPDLC-user.

3.3.10 CPDLC-provider-ab rt servic

3.3.10.1 General

3.3.10.1.1 This service provides the capability for the CPDLC service provider to inform its active users that it can no longer provide the CPDLC service. Messages in transit may be lost during this operation.

3.3.10.1.2 The CPDLC-provider-abort service shall contain the primitives and parameters as presented in Table 3-9.

Table 3-9. CPDLC-provider-abort service parameters

Parameter name	*Ind*
Reason	M

3.3.10.2 Reason parameter

3.3.10.2.1 This parameter identifies the reason for the abort.

3.3.10.2.2 The *Reason* parameter shall conform to the ASN.1 abstract syntax CPDLCProviderAbortReason.

3.4 FORMAL DEFINITIONS OF MESSAGES

3.4.1 Encoding/decoding rules

3.4.1.1 A CPDLC-air-ASE shall be capable of encoding AircraftPDUs APDUs and decoding GroundPDUs APDUs as defined in the ASN.1 module CPDLCAPDUsVersion1 specified in 3.4.2.

Note.— The ICUplinkMessage and ICDownlinkMessage ASN.1 types both contain an IC field and the EncodedCPDLCMessage type, which both resolve to a BIT STRING type. Even though this BIT STRING is specified as containing a PER-encoded ATCUplinkMessage or a PER-encoded ATCDownlinkMessage, there is no requirement on the CPDLC-air-ASE to verify this. The CPDLC-air-user is responsible for encoding a valid ATCDownlinkMessage and for verifying the correctness of a received ATCUplinkMessage.

3.4.1.2 The CPDLC-air-user shall be capable of encoding ATCDownlinkMessage types and decoding ATCUplinkMessage types, as defined in the ASN.1 module CPDLCMessageSetVersion1 specified in 3.4.3.

3.4.1.3 A CPDLC-ground-ASE shall be capable of encoding GroundPDUs APDUs and decoding AircraftPDUs APDUs as defined in the ASN.1 module CPDLCAPDUsVersion1 specified in 3.4.2.

Note 1.— The ICUplinkMessage and ICDownlinkMessage ASN.1 types both contain an IC field and the CPDLCMessage type, which both resolve to a BIT STRING type. Even though this BIT STRING is specified as containing a PER-encoded ATCUplinkMessage or a PER-encoded ATCDownlinkMessage, there is no requirement on the CPDLC-ground-ASE to verify this. The CPDLC-ground-user is responsible for encoding a valid ATCUplinkMessage and for verifying the correctness of a received ATCDownlinkMessage.

Note 2.— The ForwardMessage ASN.1 type contains the uplink or downlink CPDLC message element data type, which both resolve to a BIT STRING type. Even though this BIT STRING is specified as containing a PER-encoded ATCUplinkMessageData or a PER-encoded ATCDownlinkMessageData, there is no requirement on the CPDLC-ground-ASE to verify this. The CPDLC-ground-user is responsible for encoding and verifying the correctness of valid ATCDownlinkMessageData and ATCUplinkMessageData.

3.4.1.4 The CPDLC-ground-user shall be capable of encoding ATCUplinkMessage types and decoding ATCDownlinkMessage types as defined in the ASN.1 module CPDLCMessageSetVersion1 specified in 3.4.3.

3.4.1.5 The CPDLC-ground-user shall be capable of encoding and decoding ATCUplinkMessageData and ATCDownlinkMessageData types as defined in the ASN.1 module CPDLCMessageSetVersion1 specified in 3.4.3.

3.4.2 CPDLC ASN.1 abstract syntax

The abstract syntax of the CPDLC protocol data units shall comply with the description contained in the ASN.1 module CPDLCAPDUsVersion1 (conforming to ISO/IEC 8824), as defined hereinafter:

```
CPDLCAPDUsVersion1 DEFINITIONS::=

BEGIN

IMPORTS
        DateTimeGroup,
        AircraftFlightIdentification,
        AircraftAddress
        FROM CPDLCMessageSetVersion1;

-- ------------------------------------------------------------------------
-- Ground Generated Messages - Top level
-- ------------------------------------------------------------------------
GroundPDUs ::= CHOICE
        {
        abortUser               [0]     CPDLCUserAbortReason,
        abortProvider           [1]     CPDLCProviderAbortReason,
        startup                 [2]     ICUplinkMessage,
        send                    [3]     ICUplinkMessage,
        forward                 [4]     ATCForwardMessage,
        forwardresponse         [5]     ATCForwardResponse,
        ...
        }


ICUplinkMessage<blank> ::= SEQUENCE
        {
        algorithmIdentifier     [0]     AlgorithmIdentifier OPTIONAL,
        embeddedMessage         [1]     EncodedCPDLCMessage OPTIONAL,
                                        -- PER encoded ATCUplinkMessage
                                        -- (see Module CPDLCMessageSetVersion1)
```

```
        integrityCheck          [2]     BIT STRING,
        ...
        }

ATCForwardMessage ::= SEQUENCE
        {
        forwardHeader           ForwardHeader,
        forwardMessage          ForwardMessage
        }

ForwardHeader ::= SEQUENCE
        {
        dateTime                DateTimeGroup,
        aircraftID              AircraftFlightIdentification,
        aircraftAddress         AircraftAddress
        }

ForwardMessage ::= CHOICE
        {
        upElementIDs            [0]     BIT STRING,
                                        --PER encoded ATCUplinkMessageData,
                                        -- (see Module CPDLCMessageSetVersion1)
        downElementIDs          [1]     BIT STRING
                                        --PER encoded ATCDownlinkMessageData,
                                        -- (see Module CPDLCMessageSetVersion1)
        }

ATCForwardResponse ::= ENUMERATED
        {
        success                 (0),
        service-not-supported   (1),
        version-not-equal       (2),
        ...
        }


-- ------------------------------------------------------------------------
-- Aircraft Generated Messages - Top level
-- ------------------------------------------------------------------------

AircraftPDUs::= CHOICE
        {
        abortUser               [0]     CPDLCUserAbortReason,
        abortProvider           [1]     CPDLCProviderAbortReason,
        startdown               [2]     StartDownMessage,
        send                    [3]     ICDownlinkMessage,
        ...
        }
```

```
StartDownMessage ::= SEQUENCE
        {
        mode                        Mode DEFAULT cpdlc,
        startDownlinkMessage        ICDownlinkMessage
        }

Mode ::= ENUMERATED
        {
        cpdlc                (0),
        dsc                  (1)
        }


ICDownlinkMessage<blank>::= SEQUENCE
        {
        algorithmIdentifier     [0]     AlgorithmIdentifier OPTIONAL,
        embeddedMessage         [1]     EncodedCPDLCMessage OPTIONAL,
                                        --PER encoded ATCDownlinkMessage,
                                        -- (see Module CPDLCMessageSetVersion1)
        integrityCheck          [2]     BIT STRING,
        ...
        }

-- ------------------------------------------------------------------------
-- Uplink and Downlink messages - Common Elements
-- ------------------------------------------------------------------------
AlgorithmIdentifier ::= RELATIVE-OID --root is {icao-arc atn-algorithms(9)}
                                -- see Doc 9880, Part IV, for OID root assignment

EncodedCPDLCMessage ::= BIT STRING

CPDLCUserAbortReason ::= ENUMERATED
                {
                undefined                                   (0),
                no-message-identification-numbers-available (1),
                duplicate-message-identification-numbers    (2),
                no-longer-next-data-authority               (3),
                current-data-authority-abort                (4),
                commanded-termination                       (5),
                invalid-response                            (6),
                time-out-of-synchronization                 (7),
                unknown-integrity-check                     (8),
                validation-failure                          (9),
                unable-to-decode-message                    (10),
                invalid-pdu                                 (11),
                invalid-CPDLC-message                       (12),
                ...
                }
```

```
CPDLCProviderAbortReason ::= ENUMERATED
        {
        timer-expired                   (0),
        undefined-error                 (1),
        invalid-PDU                     (2),
        protocol-error                  (3),
        communication-service-error     (4),
        communication-service-failure   (5),
        invalid-QOS-parameter           (6),
        expected-PDU-missing            (7),
        ...
        }

END
```

3.4.3 CPDLC message ASN.1 abstract syntax

The abstract syntax of CPDLC messages shall comply with the description contained in the ASN.1 module CPDLCMessageSetVersion1 (conforming to ISO/IEC 8824), as defined hereinafter.

Note.— The object identifier "cpdlc-message-AS-v1", as defined below, identifies the CPDLC Message Set Version 1:

cpdlc-message-AS-v1 ::= OBJECT IDENTIFIER {iso(1) identified organization(3) icao(27) user message abstract syntax(10) cpdlc(1) version1(1)

```
CPDLCMessageSetVersion1 DEFINITIONS::=

BEGIN

ATCUplinkMessage ::= SEQUENCE
        {
        header                  ATCMessageHeader,
        messageData             ATCUplinkMessageData
        }

ATCUplinkMessageData ::= SEQUENCE
        {
        elementIds              SEQUENCE SIZE (1..5) OF ATCUplinkMsgElementId,
        constrainedData         SEQUENCE
        {
        routeClearanceData      SEQUENCE SIZE (1..2) OF RouteClearance OPTIONAL,
        ...
        }
        OPTIONAL
        }
```

```
ATCDownlinkMessage ::= SEQUENCE
        {
        header                          ATCMessageHeader,
        messageData                     ATCDownlinkMessageData
        }

ATCDownlinkMessageData ::= SEQUENCE
        {
        elementIds                      SEQUENCE SIZE (1..5) OF ATCDownlinkMsgElementId,
        constrainedData                 SEQUENCE
        {
        routeClearanceData              SEQUENCE SIZE (1..2) OF RouteClearance OPTIONAL,
        ...
        }
        OPTIONAL
        }

-- ------------------------------------------------------------------------
-- Uplink and Downlink messages - Common Elements
-- ------------------------------------------------------------------------

ATCMessageHeader ::= SEQUENCE
        {
        messageIdNumber         [0]     MsgIdentificationNumber,
        messageRefNumber        [1]     MsgReferenceNumber      OPTIONAL,
        dateTime                [2]     DateTimeGroup,
        logicalAck              [3]     LogicalAck              DEFAULT notRequired
        }

MsgIdentificationNumber ::= INTEGER (0..63)

MsgReferenceNumber ::= INTEGER (0..63)


LogicalAck ::= ENUMERATED
        {
        required                        (0),
        notRequired                     (1)
        }
-- ------------------------------------------------------------------------
-- Uplink message element
-- ------------------------------------------------------------------------

ATCUplinkMsgElementId ::= CHOICE
        {
--      UNABLE                                  Urg(N)/Alr(M)/Resp(N)
        uM0NULL                                 [0] NULL,

--      STANDBY                                 Urg(N)/Alr(L)/Resp(N)
        uM1NULL                                 [1] NULL,
```

--	REQUEST DEFERRED uM2NULL	Urg(N)/Alr(L)/Resp(N) [2] NULL,
--	ROGER uM3NULL	Urg(N)/Alr(L)/Resp(N) [3] NULL,
--	AFFIRM uM4NULL	Urg(N)/Alr(L)/Resp(N) [4] NULL,
--	NEGATIVE uM5NULL	Urg(N)/Alr(L)/Resp(N) [5] NULL,
--	EXPECT [level] uM6Level	Urg(L)/Alr(L)/Resp(R) [6] Level,
--	EXPECT CLIMB AT [time] uM7Time	Urg(L)/Alr(L)/Resp(R) [7] Time,
--	EXPECT CLIMB AT [position] uM8Position	Urg(L)/Alr(L)/Resp(R) [8] Position,
--	EXPECT DESCENT AT [time] uM9Time	Urg(L)/Alr(L)/Resp(R) [9] Time,
--	EXPECT DESCENT AT [position] uM10Position	Urg(L)/Alr(L)/Resp(R) [10] Position,
--	EXPECT CRUISE CLIMB AT [time] uM11Time	Urg(L)/Alr(L)/Resp(R) [11] Time,
--	EXPECT CRUISE CLIMB AT [position] uM12Position	Urg(L)/Alr(L)/Resp(R) [12] Position,
--	AT [time] EXPECT CLIMB TO [level] uM13TimeLevel	Urg(L)/Alr(L)/Resp(R) [13] TimeLevel,
--	AT [position] EXPECT CLIMB TO [level] uM14PositionLevel	Urg(L)/Alr(L)/Resp(R) [14] PositionLevel,
--	AT [time] EXPECT DESCENT TO [level] uM15TimeLevel	Urg(L)/Alr(L)/Resp(R) [15] TimeLevel,
--	AT [position] EXPECT DESCENT TO [level] uM16PositionLevel	Urg(L)/Alr(L)/Resp(R) [16] PositionLevel,
--	AT [time] EXPECT CRUISE CLIMB TO [level] uM17TimeLevel	Urg(L)/Alr(L)/Resp(R) [17] TimeLevel,
--	AT [position] EXPECT CRUISE CLIMB TO [level] uM18PositionLevel	Urg(L)/Alr(L)/Resp(R) [18] PositionLevel,

--	MAINTAIN [level] uM19Level	Urg(N)/Alr(M)/Resp(W/U) [19] Level,
--	CLIMB TO [level] uM20Level	Urg(N)/Alr(M)/Resp(W/U) [20] Level,
--	AT [time] CLIMB TO [level] uM21TimeLevel	Urg(N)/Alr(M)/Resp(W/U) [21] TimeLevel,
--	AT [position] CLIMB TO [level] uM22PositionLevel	Urg(N)/Alr(M)/Resp(W/U) [22] PositionLevel,
--	DESCEND TO [level] uM23Level	Urg(N)/Alr(M)/Resp(W/U) [23] Level,
--	AT [time] DESCEND TO [level] uM24TimeLevel	Urg(N)/Alr(M)/Resp(W/U) [24] TimeLevel,
--	AT [position] DESCEND TO [level] uM25PositionLevel	Urg(N)/Alr(M)/Resp(W/U) [25] PositionLevel,
--	CLIMB TO REACH [level] BY [time] uM26LevelTime	Urg(N)/Alr(M)/Resp(W/U) [26] LevelTime,
--	CLIMB TO REACH [level] BY [position] uM27LevelPosition	Urg(N)/Alr(M)/Resp(W/U) [27] LevelPosition,
--	DESCEND TO REACH [level] BY [time] uM28LevelTime	Urg(N)/Alr(M)/Resp(W/U) [28] LevelTime,
--	DESCEND TO REACH [level] BY [position] uM29LevelPosition	Urg(N)/Alr(M)/Resp(W/U) [29] LevelPosition,
--	MAINTAIN BLOCK [level] TO [level] uM30LevelLevel	Urg(N)/Alr(M)/Resp(W/U) [30] LevelLevel,
--	CLIMB TO AND MAINTAIN BLOCK [level] TO [level] -- uM31LevelLevel	Urg(N)/Alr(M)/Resp(W/U) [31] LevelLevel,
--	DESCEND TO AND MAINTAIN BLOCK [level] TO [level] -- uM32LevelLevel	Urg(N)/Alr(M)/Resp(W/U) [32] LevelLevel,
--	Reserved uM33NULL	Urg(L)/Alr(L)/Resp(Y) [33] NULL,
--	CRUISE CLIMB TO [level] uM34Level	Urg(N)/Alr(M)/Resp(W/U) [34] Level,
--	CRUISE CLIMB ABOVE [level] uM35Level	Urg(N)/Alr(M)/Resp(W/U) [35] Level,