Consultative Committee for Space Data Systems

RECOMMENDATION FOR SPACE DATA SYSTEMS STANDARDS

ADVANCED ORBITING SYSTEMS, NETWORKS AND DATA LINKS:

FORMAL SPECIFICATION OF THE VCLC SERVICE AND PROTOCOL

ADDENDUM TO CCSDS 701.0-B-2

CCSDS 705.3-B-1

BLUE BOOK

May 1994

AUTHORITY

Issue:	Blue Book, Issue 1
Date:	May 1994
Location:	Villafranca, Spain

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS Recommendations is detailed in reference [1], and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This Recommendation is published and maintained by:

CCSDS Secretariat
Program Integration Division (Code OI)
National Aeronautics and Space Administration
Washington, DC 20546, USA

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of member space Agencies. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed RECOMMENDATIONS and are not considered binding on any Agency.

This RECOMMENDATION is issued by, and represents the consensus of, the CCSDS Plenary body. Agency endorsement of this RECOMMENDATION is entirely voluntary. Endorsement, however, indicates the following understandings:

- Whenever an Agency establishes a CCSDS-related STANDARD, this STANDARD will be in accord with the relevant RECOMMENDATION. Establishing such a STANDARD does not preclude other provisions which an Agency may develop.

- Whenever an Agency establishes a CCSDS-related STANDARD, the Agency will provide other CCSDS member Agencies with the following information:

  -- The STANDARD itself.

  -- The anticipated date of initial operational capability.

  -- The anticipated duration of operational service.

- Specific service arrangements shall be made via memoranda of agreement. Neither this RECOMMENDATION nor any ensuing STANDARD is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this Recommendation will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or, (3) be retired or cancelled.

In those instances when a new version of a RECOMMENDATION is issued, existing CCSDS-related Agency standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each Agency to determine when such standards or implementations are to be modified. Each Agency is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommendation.

FOREWORD

This document, which is a technical Recommendation prepared by the Consultative Committee for Space Data Systems (CCSDS), is intended for use by participating space Agencies in their development of 'Advanced Orbiting Systems'.

This Recommendation, written using the ISO Formal Description Technique LOTOS, contains a formal specification of the VCLC Layer Protocol and Service, described in Natural Language in reference [2]. Annex A contains a set of tests, also written using LOTOS, which specify the required behaviour of the VCLC Layer Protocol and Service under certain control and input conditions.

The Abstract Data Types used within this document are given in full in reference [4], and the rationale behind the production of this formal specification is given in reference [7].

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommendation is therefore subject to CCSDS document management and change control procedures which are defined in reference [1].

At time of publication, the active Member and Observer Agencies of the CCSDS were

Member Agencies

- British National Space Centre (BNSC)/United Kingdom.
- Canadian Space Agency (CSA)/Canada.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centre National d'Etudes Spatiales (CNES)/France.
- Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- National Aeronautics and Space Administration (NASA HQ)/USA.
- National Space Development Agency of Japan (NASDA)/Japan.

Observer Agencies

- Australian Space Office (ASO)/Australia.
- Austrian Space Agency (ASA)/Austria.
- Belgian Science Policy Office (SPO)/Belgium.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Space Technology (CAST)/China.
- Communications Research Laboratory (CRL)/Japan.
- Danish Space Research Institute (DSRI)/Denmark.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Industry Canada/Communications Research Center (CRC)/Canada.
- Institute of Space and Astronautical Science (ISAS)/Japan.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Ministry of Communications (MOC)/Israel.
- National Oceanic & Atmospheric Administration (NOAA)/USA.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 705.3-B-1	Recommendation for Space Data Systems Standards—Advanced Orbiting Systems, Networks and Data Links: Formal Specification of the VCLC Service and Protocol—Addendum to CCSDS 701.0-B-2, Issue 1	May 1994	Original Issue

CONTENTS

Sections Page

REFERENCES ..... **vii**

1 PURPOSE AND SCOPE **1-1**

2 VIRTUAL CHANNEL LINK CONTROL PROTOCOL **2-1**

2.1 INTRODUCTION 2-2
2.2 SPECIFICATION ARCHITECTURE 2-2

3 VIRTUAL CHANNEL LINK CONTROL SERVICE 3-1

ANNEX A VCLC PROTOCOL AND SERVICE TESTS **A-1**

Figures

2-1 Specification Internal Breakdown 2-2
2-2 Encapsulation Service Interfaces 2-7
2-3 Bitstream Procedure Interfaces 2-21
3-1 Encapsulation Service Interfaces 3-1

REFERENCES

[1] *Procedures Manual for the Consultative Committee for Space Data Systems*. CCSDS A00.0-Y-6. Yellow Book. Issue 6. Washington, D.C.: CCSDS, May 1994 or later issue.

[2] *Advanced Orbiting Systems, Networks and Data Links: Architectural Specification*. Recommendation for Space Data Systems Standards, CCSDS 701.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, November 1992 or later issue.

[3] *Information Processing Systems—Open Systems Interconnection—LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. ISO 8807. Issue 1. Geneva: ISO, 1989.

[4] *Advanced Orbiting Systems, Networks and Data Links: Abstract Data Type Library—Addendum to CCSDS 701.0-B-2.* Recommendation for Space Data Systems Standards, CCSDS 705.1-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1994 or later issue.

[5] *Advanced Orbiting Systems, Networks and Data Links: Formal Specification of the Path Service and Protocol—Addendum to CCSDS 701.0-B-2*. Recommendation for Space Data Systems Standards, CCSDS 705.2-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1994 or later issue.

[6] *Advanced Orbiting Systems, Networks and Data Links: Formal Specification of the VCA Service and Protocol—Addendum to CCSDS 701.0-B-2*. Recommendation for Space Data Systems Standards, CCSDS 705.4-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1994 or later issue.

[7] *Advanced Orbiting Systems, Networks and Data Links: Formal Definition of CPN Protocols, Methodology and Approach*. Report Concerning Space Data Systems Standards, CCSDS 705.0-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, October 1993 or later issue.

[8] *Advanced Orbiting Systems, Networks and Data Links: Summary of Concept, Rationale and Performance*. Report Concerning Space Data Systems Standards, CCSDS 700.0-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, November 1992 or later issue.

1 PURPOSE AND SCOPE

This document provides formal specifications of the Consultative Committee for Space Data Systems (CCSDS) Advanced Orbiting Systems (AOS) VCLC service and protocol[1] using the ISO LOTOS formal description technique (refer to reference [3]). These formal specifications are not intended as replacements for the natural-language specifications provided in the AOS Blue Book (reference [2]), but as unambiguous expressions of those specifications, which may be used to clarify any problem areas.

This document is one of four CCSDS Recommendations that provide LOTOS specifications for the suite of AOS services and protocols (see references [4] through [6]). The relationship between the main AOS Recommendation and the four LOTOS Specifications is shown below; the numbers to the right are the CCSDS document references for the Recommendations containing the LOTOS Specifications.

ADT Library	705.1
Path Service	705.2
Path Protocol	705.2
VCLC Service	705.3
VCLC Protocol	705.3
VCA Service	705.4
VCA Protocol	705.4

A supporting CCSDS Report (reference [7]) contains the rationale, methodology, and approach used to prepare the LOTOS specifications.

These documents are expected to be of use primarily to the technical experts responsible for the design, configuration, and testing of AOS implementations; a basic knowledge of LOTOS is required to understand the formal specifications. Other users of the AOS services should consult the main AOS Recommendation and the companion CCSDS Report (references [2] and [8]).

---

[1]The natural-language specifications for the VCLC service and protocol are contained in the AOS Blue Book, CCSDS 701.0-B-2, reference [2].

2 VIRTUAL CHANNEL LINK CONTROL PROTOCOL

```
specification VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                            (EncapDataLossEnabled : Bool,
                             BitDataLossEnabled : Bool,
                             PType : PacketType,
                             MuxFill : OctetString,
                             BitFill : BitString) : noexit
```

This specification represents the functional requirements of the Virtual Channel Link Control (VCLC) Sublayer protocols, for use in the Space Link Subnetwork (SLS), as described in *Advanced Orbiting Systems, Networks and Data Links: Architectural Specification* (reference [2]).

For the purpose of this LOTOS specification, values are assumed to be available for the parameters PType, MuxFill and BitFill. However, it is not intended that an implementation be constrained to use fixed values in these fields.

2.1 INTRODUCTION

The CCSDS Virtual Channel Link Control service provides transfer of 'bitstream' data or octet-aligned 'packetized' data across a Virtual Channel. Packetized data may be multiplexed together from a number of sources onto a single Virtual Channel, and may be formatted either as CCSDS Version-1 Packets, or as ISO 8473 Packets which will be encapsulated within CCSDS Version-1 Packets by the VCLC service itself. This concept of 'encapsulation' may be extended to other Packet formats in the future; this specification deals only with the encapsulation of ISO 8473. Bitstream data are placed into Virtual Channels dedicated to individual sources; the VCLC service does not provide for the multiplexing of bitstream data.

The Virtual Channel Link Control service uses the Virtual Channel Access service to place the data associated with Virtual Channels into formatted Virtual Channel Data Units.

2.2 SPECIFICATION ARCHITECTURE

The LOTOS specification is broken down internally as shown in Figure 2-1. For clarity, each service interface has been shown twice, one instance showing the associated service indication, the other showing the service request. Each of these interfaces is described in more detail below.

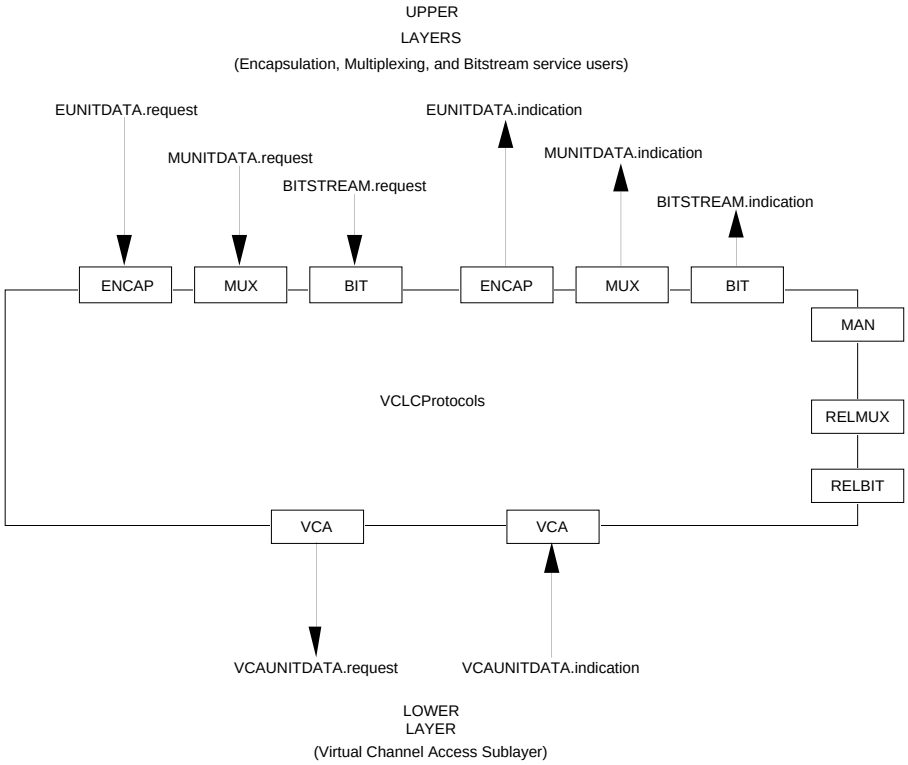

Figure 2-1: Specification Internal Breakdown

[encap] represents the interface between a VCLC User and an Encapsulation Procedure. A service primitive E_UNITDATA.request is passed to the VCLC sublayer to request that an E_SDU be encapsulated and multiplexed into the specified Virtual Channel and sent. The service primitive takes the form of an E_SDU, VCDU-ID, PCID. For de-Encapsulation, an E_UNITDATA.indication is employed. An E_UNITDATA.indication is passed from the VCLC sublayer to indicate the arrival of an E_SDU.

[enmux] represents the internal VCLC interface between the Encapsulation Procedure and the Multiplexing Procedure. It serves the same purpose as the [mux] interface except that it is not exposed to outside users and remains internal to the VCLC only.

[mux] represents the interface between a Multiplexing Procedure and a VCLC User, through which the Multiplexing Procedure sends and receives M_SDUs. An M_UNITDATA.request is used to send data to a Multiplexing Procedure and an M_UNITDATA.indication is used to send data from a Multiplexing Procedure to the upper layer.

[bit] represents the interface between a Bitstream Procedure and a Bitstream User, through which the Bitstream User sends and receives Bitstream data. The assumption made in this specification is that data is received one bit at a time. This does not constrain an implementation; it is done for flexibility. A BITSTREAM.request is used to send data to a Bitstream Procedure and a BITSTREAM.indication is used to send data from a Bitstream Procedure to a Bitstream User.

[vca] represents the interface between the VCLC and the VCA, through which a Bitstream Procedure or a Multiplexing Procedure sends and receives data. A VCA_UNITDATA.request is used to send data to the VCA and a VCA_UNITDATA.indication is received from the VCA.

[relmux] represents an interface to the Multiplexing Procedure, which generates events that the Multiplexing Procedure will respond to by sending a PDU,

and [relbit] represents an interface to the Bitstream Procedure, which generates events that the Bitstream Procedure will respond to by sending a PDU.

[man] represents the interface with management for configuring the proper protocol layers, sub-layers, or procedures. Since the specifications are written to require a separate instantiation for each different stream of data (each Multiplexing Procedure handles only one VCDU-ID, for instance), it is data coming in from this interface that does the instantiation and contains the initial data required.

Note – The [man] gate does not represent a true management interface; functions such as stopping, restarting and interrogating the protocol entities have not been specified. The [man] gate interaction only performs instantiation of protocol entities. The management interface shown here is not intended to constrain an implementation and does not reflect the AOS Blue Book (reference [2]). It is designed to allow a minimal set of management actions to be performed in order to execute the protocol behaviour.

ADTs found in the Library:

```
library
        CCSDSPacket, PathID, SecondaryHeaderIndicator, DataLossIndicator,
        ESDULossFlag, MPDU, BPDU, BitstreamDataLossFlag, BitStreamDataPointer,
        VCDUID, VCDULossFlag, Boolean
endlib
```

The ADT, ServiceType, is specific to the LOTOS specification of the VCLC and is not intended to constrain any implementation.

```
type    ServiceType is Boolean
sorts   ServiceType
opns    Mux, DeMux                      : -> ServiceType
        Encap, DeEncap                  : -> ServiceType
        Bit, DeBit                      : -> ServiceType
        _eq_                            : ServiceType, ServiceType -> Bool
        _ne_                            : ServiceType, ServiceType -> Bool
eqns    forall st1, st2 : ServiceType

        ofsort Bool

        st1 eq st1                      = true                  ;
        st1 ne st1                      = false                 ;
            st1 eq st2 =>
        st1 eq st2                      = true                  ;
            st1 eq st2 =>
        st2 eq st1                      = true                  ;
            st1 eq st2 =>
        st1 ne st2                      = false                 ;
            st1 ne st2 =>
        st2 ne st1                      = true                  ;

endtype
```

```
behaviour

VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
              (EncapDataLossEnabled,
               BitDataLossEnabled,
               PType,
               MuxFill,
               BitFill)
where

process VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                      (EncapDataLossEnabled : Bool,
                       BitDataLossEnabled : Bool,
                       PType : PacketType,
                       MuxFill : OctetString,
                       BitFill : BitString) : noexit :=
(
     hide enmux in
     (
          EncapsulationProcedures [encap, man, enmux]
                                  (EncapDataLossEnabled, PType)
          |[enmux]|
          MultiplexingProcedures [enmux, mux, man, relmux, vca]
                                 (MuxFill, PType)
     )
     |||
     BitStreamProcedures [bit, man, relbit, vca]
                         (BitDataLossEnabled, BitFill)
)

where
```

In the following description, a value of all zeroes for the initial PacketSequence Count is chosen for convenience only in both ENACP and DEENCAP. An implementation may chose to initialise this count with some other value.

```
process EncapsulationProcedures [encap, man, enmux]
                                (generateDataLossFlags : Bool,
                                 pType : PacketType) : noexit :=
(
     man ? ServiceIndicator : ServiceType
         ? EncapVCDUID : VCDUID
         ? EncapPCID : APID
         ? MaxESDUSize : Nat
               [ServiceIndicator eq Encap] ;

          (
          ENCAP  [encap, enmux]
                 (PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
                  pType,
                  EncapVCDUID,
                  EncapPCID,
                  MaxESDUSize)
          |||
          EncapsulationProcedures [encap, man, enmux]
                                  (generateDataLossFlags,
                                   pType)
          )

        []

        man ? ServiceIndicator : ServiceType
            ? EncapVCDUID : VCDUID
            ? EncapPCID : APID
               [ServiceIndicator eq DeEncap] ;

          (
          DEENCAP[encap, enmux]
             (PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
              generateDataLossFlags,
              EncapVCDUID,
              EncapPCID,
              True)

          |||
          EncapsulationProcedures [encap, man, enmux]
                                  (generatedataLossFlags,
                                   pType)
          )
)

where
```

This specification provides the functional requirements for the Encapsulation service within the VCLC sublayer, for use in the Space Link Subnetwork (SLS), as described in the AOS Blue Book (reference [2]), Chapter 5.

The Encapsulation service provides transfer of non-CCSDS-structured delimited, octet-aligned Encapsulation SDUs (E_SDUs) across the Space Link Subnet by encapsulating them within Encapsulation Protocol Data Units (CCSDSPackets). The CCSDSPackets use the CCSDS Packet structure; i.e., a CCSDS Primary Header is appended to the E_SDU for multiplexing within a virtual channel. The CCSDS Packet structure also incorporates the use of an optional CCSDS Secondary Header but this option will not be included as part of this version of the Encapsulation service specified herein. The service can permit E_SDUs to have any format, and to be of any length which is an integral number of octets, up to 65,536 octets maximum. In the reverse procedure, a CCSDSPacket may be de-encapsulated by extracting the E_SDU from the CCSDS Packet Structure. Both aspects are addressed within this specification.

The Packet Channel Identifiers (PCIDs) 2031 to 2046 are reserved for use by the Encapsulation service. The current specification allows only for the encapsulation of ISO 8473 packets. These are assigned the PCID number 2046.

The Encapsulation Service Procedure is specified with two basic interfaces: a VCLC user upper-layer interface (encap) and a lower-layer interface with the Multiplexing Service Procedure (enmux). Refer to Figure 2-2.

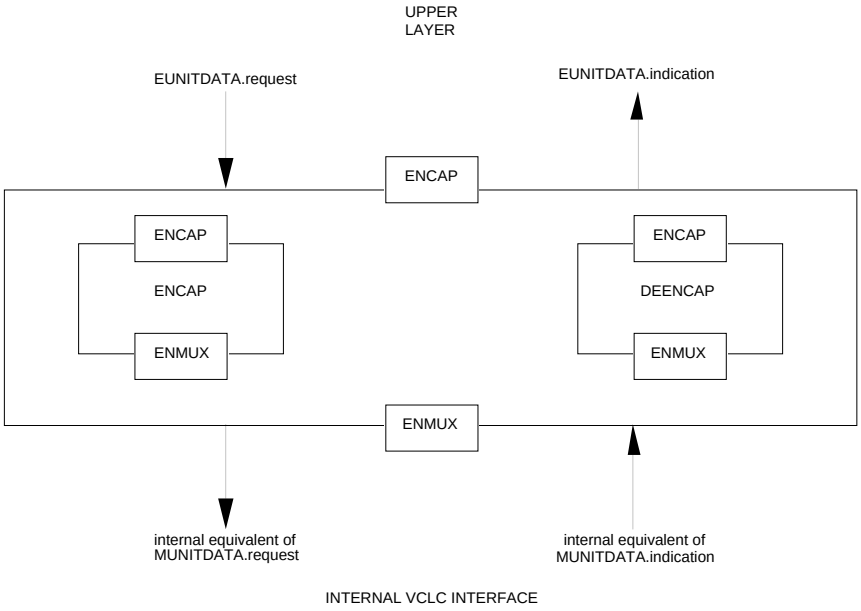

Figure 2-2: Encapsulation Service Interfaces

```
process ENCAP [encap, enmux]
          (PSCT : PacketSequenceCount,
           pType  : PacketType,
           EncapVCDUID : VCDUID,
           EncapPCID : APID,
           MaxESDUSize : Nat) : noexit :=
      (
      encap ! EncapVCDUID
            ! EncapPCID
            ? ESDU : OctetString [(LengthOf(ESDU) Le MaxESDUSize) And
                                  (EncapPCID Eq 8473EncapPacketAPID)] ;

      enmux ! EncapVCDUID
            ! EncapPCID
            ! MakeCCSDSPacket(MakePrimaryHeader(
                                MakePacketID(Version1,pType,
                                             SHAbsent,EncapPCID),
                                MakePacketSC(PacketSequenceUnSeg,next(PSCT)),
                                ConvertNatToPL(pred(LengthOf(ESDU)))),
                               ESDU) ;

      ENCAP [encap, enmux]
          (next(PSCT),
           pType,
           EncapVCDUID,
           EncapPCID,
           MaxESDUSize)

      )

endproc ENCAP
```

```
process DEENCAP [encap, enmux]
               (CurrentPSC : PacketSequenceCount,
                GenerateDataLossFlags : Bool,
                EncapVCDUID : VCDUID,
                EncapPCID : APID,
                FirstPacket : Bool) : noexit :=

(
     enmux ! EncapVCDUID
           ! EncapPCID
           ? EPDU : CCSDSPacket ;
     (
     [GenerateDataLossFlags eq true] ->
          (
          [(Next(CurrentPSC) eq
           GetPacketSequenceCount(GetPrimaryHeader(EPDU))) Or FirstPacket] ->
               (
               encap ! EncapVCDUID
                     ! EncapPCID
                     ! GetUserData(EPDU)
                     ! ESDUNotLost [EncapPCID Eq 8473EncapPacketAPID] ;

               DEENCAP[encap, enmux]
                      (GetPacketSequenceCount(GetPrimaryHeader(EPDU)),
                       GenerateDataLossFlags,
                       EncapVCDUID,
                       EncapPCID,
                       False)
               )
          []
          [(Next(CurrentPSC) ne
           GetPacketSequenceCount(GetPrimaryHeader(EPDU))) And
           Not(FirstPacket)] ->
               (
               encap ! EncapVCDUID
                     ! EncapPCID
                     ! GetUserData(EPDU)
                     ! ESDULost [EncapPCID Eq 8473EncapPacketAPID] ;

               DEENCAP[encap, enmux]
                      (GetPacketSequenceCount(GetPrimaryHeader(EPDU)),
                       GenerateDataLossFlags,
                       EncapVCDUID,
                       EncapPCID,
                       False)
               )
          )
     []
     [GenerateDataLossFlags Eq False] ->
          (
          encap ! EncapVCDUID
                ! EncapPCID
                ! GetUserData(EPDU) [EncapPCID Eq 8473EncapPacketAPID] ;
          DEENCAP[encap, enmux]
                 (GetPacketSequenceCount(GetPrimaryHeader(EPDU)),
                  GenerateDataLossFlags,
                  EncapVCDUID,
                  EncapPCID,
                  False)
          )
     )
)

endproc DEENCAP

endproc EncapsulationProcedures
```

```
 process MultiplexingProcedures [enmux, mux, man, relmux, vca]
                               (MuxFill : OctetString,
                                pType : PacketType) : noexit :=
(
     man ? ServiceIndicator : ServiceType
         ? PZSize : Nat
         ? MuxVCDUID : VCDUID
         ? MaxMSDUSize : Nat
              [ServiceIndicator Eq Mux] ;
     (
          Multiplexing [mux, enmux, relmux, vca]
                       (MuxVCDUID,
                        MakeMPDU(MakeMPDUHeader(MPDUSpare, NullFHP), NullOS),
                        PZSize,
                        pType,
                        MuxFill,
                        MaxMSDUSize)
          |||
          MultiplexingProcedures [enmux, mux, man, relmux, vca]
                                 (MuxFill, pType)
     )
     []
     man ? ServiceIndicator : ServiceType
         ? MuxVCDUID : VCDUID
              [ServiceIndicator eq DeMux] ;

     (
          DeMultiplexing [enmux, mux, vca]
                         (MuxVCDUID,
                          NullOS)

          |||
          MultiplexingProcedures [enmux, mux, man, relmux, vca]
                                 (MuxFill, pType)
     )
)

where
```

```
process Multiplexing [mux, enmux, relmux, vca]
                     (VCDUID : VCDUID,
                      MPDU : MPDU,
                      PZSize : Nat,
                      PType : PacketType,
                      MuxFill : OctetString,
                      MaxMSDUSize : Nat) : noexit :=
(
     (
          mux ! VCDUID
              ? PCID : APID
              ? MSDU : CCSDSPacket
              [(TotalLengthOfPacket(MSDU) Le MaxMSDUSize) And
               ValidPacketLength(MSDU) And
               (GetVersion(GetPrimaryHeader(MSDU)) Eq Version1) And
               UserAPID(PCID)] ;
          AddToMPDU [vca] (VCDUID, MPDU, ConvertPktToOS(MSDU), PZSize) >>
          accept NewMPDU : MPDU in
          Multiplexing [mux, enmux, relmux, vca]
                       (VCDUID, NewMPDU, PZSize, PType, MuxFill, MaxMSDUSize)
     )
     []
     (
          enmux ! VCDUID
                ? PCID : APID
                ? EPDU : CCSDSPacket
                [(TotalLengthOfPacket(EPDU) Le MaxMSDUSize) And
                 ValidPacketLength(EPDU) And
                 (GetVersion(GetPrimaryHeader(EPDU)) Eq Version1) And
                 Not(UserAPID(PCID)) And
                 (PCID Ne FillPacketAPID)] ;
          AddToMPDU [vca] (VCDUID, MPDU, ConvertPktToOS(EPDU), PZSize) >>
          accept NewMPDU : MPDU in
          Multiplexing [mux, enmux, relmux, vca]
                       (VCDUID, NewMPDU, PZSize, PType, MuxFill, MaxMSDUSize)
     )
     []
     (
          relmux ! VCDUID ;
          (
          [GetMPDUPZ(MPDU) Eq NullOS] ->
               (
               vca ! VCDUID
                   ! ConvertMPDUToOS
                      (MakeMPDU(MakeMPDUHeader(MPDUSpare, FillFHP),
                                ConvertPktToOS(MakeFillPacket(PType,
                                                              PZSize-6,
                                                              MuxFill)))) ;
               Multiplexing [mux, enmux, relmux, vca]
                            (VCDUID,
                            MakeMPDU(MakeMPDUHeader(MPDUSpare, NullFHP),
                                     NullOS),
                            PZSize,
                            PType,
                            MuxFill,
                            MaxMSDUSize)
               )
```

```
        []
        [GetMPDUPZ(MPDU) Ne NullOS] ->
             (
             let RemainingLength : Nat = (PZSize - LengthOf(GetMPDUPZ(MPDU)))
             in
             (
             [RemainingLength Lt 7] ->
                  (
                  AddToMPDU [vca] (VCDUID,
                                   MPDU,
                                   ConvertPktToOS(MakeFillPacket(PType,
                                                                  1,
                                                                  MuxFill)),
                                   PZSize) >>
                  accept NewMPDU : MPDU in
                  Multiplexing [mux, enmux, relmux, vca]
                               (VCDUID, NewMPDU, PZSize,
                                PType, MuxFill, MaxMSDUSize)
                  )
             []
             [RemainingLength Ge 7] ->
                  (
                  vca ! VCDUID
                      ! ConvertMPDUToOS
                         (MakeMPDU(GetMPDUHeader(MPDU),
                                   Append(ConvertPktToOS
                                    (MakeFillPacket(PType,
                                                    RemainingLength-6,
                                                    MuxFill)),
                                          GetMPDUPZ(MPDU)))) ;
                  Multiplexing [mux, enmux, relmux, vca]
                               (VCDUID,
                                MakeMPDU(MakeMPDUHeader(MPDUSpare,NullFHP),
NullOS),
                                PZSize,
                                PType,
                                MuxFill,
                                MaxMSDUSize)
                  )
             )
             )
         )
    )
)

where
```

```
process AddToMPDU [vca] (VCDUID : VCDUID,
                         MPDU : MPDU,
                         NewOS : OctetString,
                         PZSize : Nat) : exit (MPDU) :=
(
let MPDUHeader : MPDUHeader = GetMPDUHeader(MPDU),
    MPDUPZ : OctetString = GetMPDUPZ(MPDU),
    PacketZoneLeft : Nat = PZSize - LengthOf(GetMPDUPZ(MPDU))
in
    (
    [PacketZoneLeft Gt LengthOf(NewOS)] ->
         exit(MakeMPDU(MPDUHeader, Append(NewOS, MPDUPZ)))
    []
    [PacketZoneLeft Eq LengthOf(NewOS)] ->
         (
         vca ! VCDUID
             ! ConvertMPDUToOS(MakeMPDU(MPDUHeader, Append(NewOS, MPDUPZ))) ;
         exit(MakeMPDU(MakeMPDUHeader(MPDUSpare, NullFHP), NullOS))
         )
    []
    [PacketZoneLeft Lt LengthOf(NewOS)] ->
        (
        vca ! VCDUID
            ! ConvertMPDUToOS(MakeMPDU(MPDUHeader,
                                       Append(RetainOctets(NewOS, PacketZoneLeft),
                                              MPDUPZ))) ;
             (
             let LeftOverOctets : OctetString = StripOctets(NewOS, PacketZoneLeft)
             in
             (
             [LengthOf(LeftOverOctets) Ge PZSize] ->
                  SegmentPacket [vca] (VCDUID,
                                       LeftOverOctets,
                                       PZSize)
             []
             [LengthOf(LeftOverOctets) Lt PZSize] ->
                  exit(MakeMPDU(MakeMPDUHeader
                                 (MPDUSpare,
                                  ConvertNatToFHP(LengthOf(LeftOverOctets))),
                                LeftOverOctets))
             )
             )
         )
    )
)

where
```

```
process SegmentPacket [vca] (VCDUID : VCDUID,
                             OctetsToSend : OctetString,
                             PZSize : Nat) : exit(MPDU) :=
(
     vca ! VCDUID
         ! ConvertMPDUToOS(MakeMPDU(MakeMPDUHeader(MPDUSpare, NoPacketHeaderFHP),
                                   RetainOctets(OctetsToSend, PZSize))) ;
     (
     let LeftOverOctets : OctetString = StripOctets(OctetsToSend, PZSize)
     in
     [LengthOf(LeftOverOctets) Ge PZSize] ->
          SegmentPacket [vca] (VCDUID,
                               LeftOverOctets,
                               PZSize)
     []
     [LengthOf(LeftOverOctets) Lt PZSize] ->
          exit(MakeMPDU(MakeMPDUHeader(MPDUSpare,
                                       ConvertNatToFHP(LengthOf(LeftOverOctets))),
                        LeftOverOctets))
     )
)
endproc SegmentPacket

endproc AddToMPDU

endproc Multiplexing
```

The process DEMULTIPLEXING extracts the M_SDUs received within M_PDUs and sends them on to the receiving Multiplexing Service User or the de-Encapsulator in the order received.

```
process DeMultiplexing [enmux, mux, vca]
                       (VirtualChannel : VCDUID,
                        PartialPacket : OctetString) : noexit :=
(
     vca ! VirtualChannel
         ? MPDU : OctetString
         ? DataLossFlag : VCDULossFlag ;
     (
     [DataLossFlag Eq VCDULost]->
          (
          ExtractFirstPacket [enmux, mux]
                             (GetFHP(GetMPDUHeader(ConvertOSToMPDU(MPDU))),
                              GetMPDUPZ(ConvertOSToMPDU(MPDU)),
                              VirtualChannel,
                              PartialPacket,
                              True)  >>
          accept RemainingOctets : OctetString in
          Demultiplexing [enmux, mux, vca]
                         (VirtualChannel,
                          RemainingOctets)
          )
     []
     [DataLossFlag Eq VCDUNotLost] ->
          (
          ExtractFirstPacket [enmux, mux]
                             (GetFHP(GetMPDUHeader(ConvertOSToMPDU(MPDU))),
                              GetMPDUPZ(ConvertOSToMPDU(MPDU)),
                              VirtualChannel,
                              PartialPacket,
                              False)  >>
          accept RemainingOctets : OctetString in
          Demultiplexing [enmux, mux, vca]
                         (VirtualChannel,
                          RemainingOctets)
          )
     )
)

where
```

```
process ExtractFirstPacket [enmux,mux]
                           (FHP : MPDUFHP,
                            DataZone : OctetString,
                            VirtualChannel : VCDUID,
                            PartialPacket : OctetString,
                            PDULoss : Bool) : exit (OctetString) :=
(
[FHP Eq FillFHP] ->
     exit (NullOS)
[]
[FHP Eq NoPacketHeaderFHP] ->
     (
     [(PartialPacket Eq NullOS) Or PDULoss] ->
          exit(NullOS)
     []
     [(PartialPacket Ne NullOS) And Not(PDULoss)] ->
          (
          let PH : PrimaryHeader = ConvertOStoPH
                                     (RetainOctets(Append(DataZone,
                                                   PartialPacket), 6)) in
               (
               [ConvertPLtoNat(GetPacketLength(PH))+7 Eq
                LengthOf(Append(DataZone, PartialPacket))] ->
                    (
                    DespatchPacket [enmux, mux]
                                   (Append(DataZone, PartialPacket),
                                    VirtualChannel) >>
                    exit (NullOS)
                    )
               []
               [ConvertPLtoNat(GetPacketLength(PH))+7 Ne
                LengthOf(Append(DataZone, PartialPacket))] ->
                    exit(Append(DataZone, PartialPacket))
               )
          )
     )
[]
[(FHP Ne FillFHP) And (FHP Ne NoPacketHeaderFHP) And (FHP Ne NullFHP)] ->
     (
     [PartialPacket Ne NullOS] ->
          (
          let PacketIndicatedLength : Nat =
                ConvertOSToNat(StripOctets(RetainOctets(Append(DataZone,
                                                               PartialPacket),6),4))+7,
              FHPIndicatedLength : Nat = LengthOf(PartialPacket)+ConvertFHPToNAT(FHP)
          in
               (
               [(PacketIndicatedLength Eq FHPIndicatedLength) And
                Not(PDULoss)] ->
                    (
                    DespatchPacket [enmux, mux]
                                   (Append(RetainOctets(DataZone,
ConvertFHPToNat(FHP)),
                                           PartialPacket),
                                    VirtualChannel) >>
                    ExtractOtherPackets [enmux, mux]
                                        (StripOctets(DataZone,
                                                     ConvertFHPToNat(FHP)),
                                         VirtualChannel)
                    )
               []
```

```
                [(PacketIndicatedLength Ne FHPIndicatedLength) Or PDULoss] ->
                     ExtractOtherPackets [enmux, mux]
                                         (StripOctets(DataZone,
                                                      ConvertFHPToNat(FHP)),
                                          VirtualChannel)
                )
           )
      []
      [PartialPacket Eq NullOS] ->
                ExtractOtherPackets [enmux, mux]
                                    (StripOctets(DataZone,
                                                 ConvertFHPToNat(FHP)),
                                     VirtualChannel)
      )
[]
[(FHP Ne FillFHP) And (FHP Ne NoPacketHeaderFHP) And (FHP Eq NullFHP)] ->
      (
      [PartialPacket Eq NullOS] ->
           (
           let PacketIndicatedLength : Nat =
              ConvertOSToNat(StripOctets(RetainOctets(DataZone,6),4))+7
           in
                (
                [PacketIndicatedLength Lt LengthOf(DataZone)] ->
                     (
                     DespatchPacket [enmux, mux]
                                    (RetainOctets(DataZone,
                                                  PacketIndicatedLength),
                                     VirtualChannel) >>
                     ExtractOtherPackets [enmux, mux]
                                         (StripOctets(DataZone,
                                                      PacketIndicatedLength),
                                          VirtualChannel)
                     )
                []
                [PacketIndicatedLength Eq LengthOf(DataZone)] ->
                     (
                     DespatchPacket [enmux, mux]
                                    (DataZone,
                                     VirtualChannel) >>
                     exit(NullOS)
                     )
                []
                [PacketIndicatedLength Gt LengthOf(DataZone)] ->
                     exit(DataZone)
                )
           )
      []
      [PartialPacket Ne NullOS] ->
                ExtractOtherPackets [enmux, mux]
                                    (DataZone,
                                     VirtualChannel)
      )
)

where
```

```
process ExtractOtherPackets [enmux, mux]
                            (Data : OctetString,
                             VirtualChannel : VCDUID) : exit (OctetString) :=
(
[LengthOf(Data) Le 6] ->
     exit(Data)
[]
[LengthOf(Data) Gt 6] ->
     (
     let PacketLength : PacketLength =
             GetPacketLength(ConvertOSToPH(RetainOctets(Data,6)))
     in
          (
          [ConvertPLToNat(PacketLength)+7 Le LengthOf(Data)]->
               (
               DespatchPacket [enmux, mux]
                              (RetainOctets(Data,
                                            ConvertPLToNat(PacketLength)+7),
                               VirtualChannel) >>
               ExtractOtherPackets [enmux, mux]
                                 (StripOctets(Data,
                                              ConvertPLToNat(PacketLength)+7),
                                  VirtualChannel)
               )
          []
          [ConvertPLToNat(PacketLength)+7 Gt LengthOf(Data)]->
               exit(Data)
          )
     )
)
endproc ExtractOtherPackets
```

```
process DespatchPacket [enmux, mux]
                       (Octets : OctetString,
                        VirtualChannel : VCDUID) : exit :=
(
let Packet : CCSDSPacket = ConvertOStoPkt(Octets) in
     (
     [ValidPacketLength(Packet) And
      (GetVersion(GetPrimaryHeader(Packet)) Eq Version1)] ->
          (
          [GetAPID(GetPrimaryHeader(Packet)) Eq FillPacketAPID] ->
               exit
          []
          [GetAPID(GetPrimaryHeader(Packet)) Eq 8473EncapPacketAPID] ->
               (
               enmux ! VirtualChannel
                     ! GetAPID(GetPrimaryHeader(Packet))
                     ! Packet ;
               exit
               )
          []
          [UserAPID(GetAPID(GetPrimaryHeader(Packet)))] ->
               (
               mux ! VirtualChannel
                   ! GetAPID(GetPrimaryHeader(Packet))
                   ! Packet ;
               exit
               )
          )
     []
     [Not(ValidPacketLength(Packet)) Or
      (GetVersion(GetPrimaryHeader(Packet)) Ne Version1)] ->
          exit
     )
)
endproc DespatchPacket

endproc ExtractFirstPacket

endproc Demultiplexing

endproc MultiplexingProcedures
```

```
process BitStreamProcedures [bit, man, relbit, vca]
                            (BitDataLossEnabled : Bool,
                             BitFill : BitString) : noexit :=
(
     man ? ServiceIndicator : ServiceType
         ? BitVCDUID : VCDUID
         ? BDZSize : Nat
           [ServiceIndicator Eq Bit];

     (
          BPDUCONSTR [bit, vca, relbit]
                     (BitVCDUID,
                      MakeBPDU(MakeBPDUHeader(BPDUSpare, NullBDP),
                               NullBS),
                      BDZSize,
                      BitFill)
          |||
          BitStreamProcedures [bit, man, relbit, vca]
                              (BitDataLossEnabled,
                               BitFill)
     )

     []

     man ? ServiceIndicator : ServiceType
         ? BitVCDUID : VCDUID
           [ServiceIndicator Eq DeBit];

     (
          BITSTREXTRACT [bit, vca]
                        (BitVCDUID, BitDataLossEnabled, False)
          |||
          BitStreamProcedures [bit, man, relbit, vca]
                              (BitDataLossEnabled,
                               BitFill)
     )

)

where
```

This specification represents the functional requirements of the Bitstream Procedures protocol, as described in the AOS Blue Book (reference [2]). Refer to Figure 2-3.

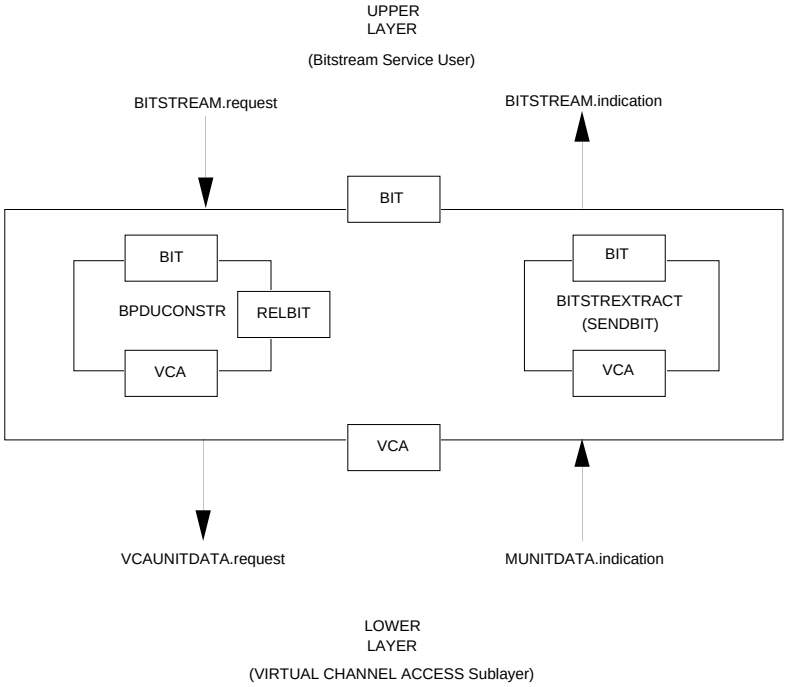

Figure 2-3: Bitstream Procedure Interfaces

An event at the [relbit] gate signals a release of a B_PDU. If the event occurs and the B_PDU is not filled, the Bitstream internal procedure will generate 'Project-specified' Fill Bits to fill the BDZ and send the B_PDU. This mechanism for setting the fill bits is not intended to constrain an implementation to a similar mechanism.

Once a B_PDU is filled, it is sent immediately. There is no queuing of B_PDUs involved in the procedure. Also, it is assumed that bits will be inserted into the B_PDU bit by bit. However, the method of generating the bitstream data need not be constrained by this; i.e., it may be continuous Bitstream, delimited Bitstream, or individual bits (see AOS Blue Book 5.3.7.5.e).

```
process BPDUCONSTR [bit, vca, relbit]
                   (VirtualChannel : VCDUID,
                    BPDU : BPDU,
                    BDZSize: Nat,
                    FillBS : BitString): noexit :=
(
     (
          bit ! VirtualChannel
              ? BSDU : Bit ;
          (
          [LengthOf(AddLSB(BSDU, GetBDZ(BPDU))) Eq BDZSize] ->
               (
                    vca ! VirtualChannel
                        ! ConvertBPDUToOS
                           (MakeBPDU(MakeBPDUHeader(BPDUSpare,NoFillDataBDP),
                                     AddLSB(BSDU, GetBDZ(BPDU)))) ;

                    BPDUCONSTR [bit, vca, relbit]
                               (VirtualChannel,
                                MakeBPDU(MakeBPDUHeader(BPDUSpare, NullBDP),
                                         NullBS),
                                BDZSize,
                                FillBS)
               )

          []
          [LengthOf(AddLSB(BSDU, GetBDZ(BPDU))) Lt BDZSize] ->
                BPDUCONSTR [bit, vca, relbit]
                   (VirtualChannel,
                    MakeBPDU(GetBPDUHeader(BPDU),
                             AddLSB(BSDU, GetBDZ(BPDU))),
                    BDZSize,
                    FillBS)
          )
     )
     []
     (
          relbit ! VirtualChannel ;
          (
          [LengthOf(GetBDZ(BPDU)) Eq 0] ->
                (
                vca ! VirtualChannel
                    ! ConvertBPDUToOS(MakeFillBPDU(BDZSize, FillBS)) ;


                BPDUCONSTR [bit, vca, relbit]
                           (VirtualChannel,
                            MakeBPDU(MakeBPDUHeader(BPDUSpare, NullBDP),
                                     NullBS),
                            BDZSize,
                            FillBS)
                )
```

```
            []
            [LengthOf(GetBDZ(BPDU)) Ne 0] ->
                  (
                  let NumberOfBits : Nat = LengthOf(GetBDZ(BPDU)),
                      NumberOfFillBits : Nat = BDZSize-LengthOf(GetBDZ(BPDU))
                  in
                       (
                       vca ! VirtualChannel
                           ! ConvertBPDUToOS
                              (MakeBPDU(MakeBPDUHeader
                                          (BPDUSpare,
                                           ConvertNatToBDP(NumberOfBits-1)),
                                       Append(MakeBitFillData(FillBS,
                                                              NumberOfFillBits),
                                              GetBDZ(BPDU)))) ;

                       BPDUCONSTR [bit, vca, relbit]
                                  (VirtualChannel,
                                   MakeBPDU(MakeBPDUHeader(BPDUSpare, NullBDP),
                                            NullBS),
                                   BDZSize,
                                   FillBS)
                       )
                  )
            )
      )
)
endproc BPDUCONSTR
```

The process BITSTREXTRACT extracts the bitstreams received within B_PDUs and sends them on as bits to the receiving bitstream user in the order received. It is an independent process and does not use the [relbit] gate as the BPDUCONSTR process does. The UnreportedDataLoss parameter is used to ensure that data loss flags delivered with FillBPDUs are delivered with the next non-Fill bitstream data.

```
process BITSTREXTRACT [bit, vca]
               (VirtualChannel : VCDUID,
                GenerateDataLossFlag : Bool,
                UnreportedDataLoss : Bool): noexit :=

(
     (
          vca  ! VirtualChannel
               ? BPDU : OctetString
               ? VCDULossFlag : VCDULossFlag ;
          (
          let BDP : BitstreamDataPointer = GetBitstreamDataPointer
                                             (GetBPDUHeader
                                               (ConvertOSToBPDU(BPDU)))
          in
               (
               [BDP Eq NoFillDataBDP] ->
                    (
                    [UnreportedDataLoss]->
                        (
                        SENDBIT [bit, vca]
                                (VirtualChannel,
                                 GetBDZ(ConvertOSToBPDU(BPDU)),
                                 GenerateDataLossFlag,
                                 BitstreamDataLost) >>
                        BITSTREXTRACT [bit, vca]
                                 (VirtualChannel,
                                  GenerateDataLossFlag,
                                  False)
                        )
                    []
                    [Not(UnreportedDataLoss)]->
                        (
                        SENDBIT [bit, vca]
                                (VirtualChannel,
                                 GetBDZ(ConvertOSToBPDU(BPDU)),
                                 GenerateDataLossFlag,
                                 ConvertVCDULFtoBitstreamLF
                                   (VCDULossFlag)) >>
                        BITSTREXTRACT [bit, vca]
                                 (VirtualChannel,
                                  GenerateDataLossFlag,
                                  False)
                        )
                    )
               []
```

```
                    [BDP Eq OnlyFillDataBDP] ->
                         (
                         [(UnreportedDataLoss) Or
                          (VCDULossFlag Eq VCDULost)]->
                              BITSTREXTRACT [bit, vca]
                                       (VirtualChannel,
                                        GenerateDataLossFlag,
                                        True)
                         []
                         [(Not(UnreportedDataLoss)) And
                          (VCDULossFlag Eq VCDUNotLost)]->
                              BITSTREXTRACT [bit, vca]
                                       (VirtualChannel,
                                        GenerateDataLossFlag,
                                        False)
                         )
                    []
                    [(BDP Ne NoFillDataBDP) And (BDP Ne OnlyFillDataBDP)] ->
                         (
                         [UnreportedDataLoss]->
                              (
                              SENDBIT [bit, vca]
                                      (VirtualChannel,
                                       RetainBits(GetBDZ
                                                  (ConvertOSToBPDU(BPDU)),
                                                   ConvertBDPToNat(BDP)+1),
                                       GenerateDataLossFlag,
                                       BitstreamDataLost) >>
                              BITSTREXTRACT [bit, vca]
                                       (VirtualChannel,
                                        GenerateDataLossFlag,
                                        False)
                              )
                         []
                         [Not(UnreportedDataLoss)]->
                              (
                              SENDBIT [bit, vca]
                                      (VirtualChannel,
                                       RetainBits(GetBDZ
                                                  (ConvertOSToBPDU(BPDU)),
                                                   ConvertBDPToNat(BDP)+1),
                                       GenerateDataLossFlag,
                                       ConvertVCDULFtoBitstreamLF
                                         (VCDULossFlag)) >>
                              BITSTREXTRACT [bit, vca]
                                       (VirtualChannel,
                                        GenerateDataLossFlag,
                                        False)
                              )
                         )
                    )
               )
          )
     )

     where
```

The process SENDBIT is used to send each bit in a Bitstream Data Zone (BDZ) to the bitstream user in the order received originally. This means sending the Most Significant Bit (MSB) of the BDZ until none are left.

```
process SENDBIT [bit, vca]
                (VirtualChannel : VCDUID,
                 BitsToSend : BitString,
                 GenerateDataLossFlag : Bool,
                 BitDataLossFlag : BitstreamDataLossFlag ) : exit :=
(
[BitsToSend Ne NullBS] ->
     (
     [GenerateDataLossFlag eq True] ->
          (
               bit ! VirtualChannel
                   ! GetMSB(BitsToSend)
                   ! BitDataLossFlag;
               SENDBIT [bit, vca]
                       (VirtualChannel,
                        RemoveMSB(BitsToSend),
                        GenerateDataLossFlag,
                        BitstreamDataNotLost )
          )
     []
     [GenerateDataLossFlag eq False] ->
          (
               bit ! VirtualChannel
                   ! GetMSB(BitsToSend) ;

               SENDBIT [bit, vca]
                       (VirtualChannel,
                        RemoveMSB(BitsToSend),
                        GenerateDataLossFlag,
                        BitDataLossFlag )
          )
     )
[]
[BitsToSend eq NullBS] ->
     exit

)

endproc SENDBIT

endproc BITSTREXTRACT

endproc BitStreamProcedures

endproc VCLCProtocols

endspec
```

3 VIRTUAL CHANNEL LINK CONTROL SERVICE

```
specification VCLCServices [encap, mux, bit, man]
                           (EncapDataLossEnabled : Bool,
                            BitDataLossEnabled : Bool) : noexit
```

This specification provides the functional requirements for the Encapsulation, Multiplexing, and Bitstream services (those services within the VCLC sublayer) for use in the Space Link Subnetwork (SLS), as described in *Advanced Orbiting Systems, Networks and Data Links: Architectural Specification* (reference [2]), Chapter 5.

The Encapsulation service provides transfer of non-CCSDS-structured delimited, octet-aligned Encapsulation SDUs (E_SDUs) across the Space Link Subnet by encapsulating them within Encapsulation Protocol Data Units (E_PDUs) across the Space Link Subnet. The E_PDUs use the CCSDS Packet structure; i.e., a CCSDS Primary Header is appended to the E_SDU for multiplexing within a virtual channel. Refer to Figure 3-1.

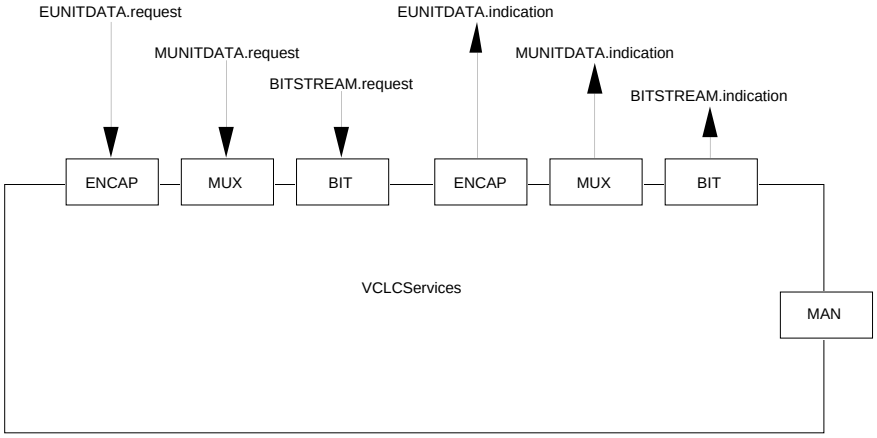

Figure 3-1: Encapsulation Service Interfaces

[encap] represents the interface between a VCLC User and an Encapsulation Procedure. A service primitive E_UNITDATA.request is passed to the VCLC sublayer to request that an E_SDU be encapsulated and multiplexed into the specified Virtual Channel and sent. The service primitive takes the form of an E_SDU, VCDU-ID, PCID. For de-Encapsulation, an E_UNITDATA.indication is employed. An E_UNITDATA.indication is passed from the VCLC sublayer to indicate the arrival of an E_SDU.

[mux] represents the interface between a Multiplexing Procedure and a VCLC User, through which the Multiplexing Procedure sends and receives M_SDUs. An M_UNITDATA.request is used to send data to a Multiplexing Procedure and an M_UNITDATA.indication is used to send data from a Multiplexing Procedure to the upper layer.

[bit] represents the interface between a Bitstream Procedure and a Bitstream User, through which the Bitstream User sends and receives Bitstream data. The assumption made in this specification is that data is received one bit at a time. This does not constrain an implementation; it is done for flexibility. A BITSTREAM.request is used to send data to a Bitstream Procedure and a BITSTREAM.indication is used to send data from a Bitstream Procedure to a Bitstream User.

ADTs found in the Library:

```
library
        CCSDSPacket, PathID, SecondaryHeaderIndicator,
        DataLossIndicator, ESDULossFlag, MPDU, BPDU,
        BitstreamDataLossFlag, VCDUID, VCDULossFlag,
        Boolean
endlib


type    ServiceType is Boolean
sorts   ServiceType
opns    Mux, DeMux                      : -> ServiceType
        Encap, DeEncap                  : -> ServiceType
        Bit, DeBit                      : -> ServiceType
        _eq_                            : ServiceType, ServiceType -> Bool
        _ne_                            : ServiceType, ServiceType -> Bool
eqns    forall st1, st2 : ServiceType

        ofsort Bool

        st1 eq st1                      = true                  ;
        st1 ne st1                      = false                 ;
            st1 eq st2 =>
        st1 eq st2                      = true                  ;
            st1 eq st2 =>
        st2 eq st1                      = true                  ;
            st1 eq st2 =>
        st1 ne st2                      = false                 ;
            st1 ne st2 =>
        st2 ne st1                      = true                  ;

endtype
```

```
behaviour

VCLCServices [encap, mux, bit, man]
             (EncapDataLossEnabled,
              BitDataLossEnabled)

where

process VCLCServices [encap, mux, bit, man]
                     (EncapDataLossEnabled : Bool,
                      BitDataLossEnabled : Bool) : noexit :=

(
     man ? ServiceIndicator : ServiceType
         ? EncapVCDUID : VCDUID
         ? EncapPCID : APID
         ? MaxESDUSize : Nat
           [ServiceIndicator Eq Encap] ;
     (
          VCLCServices [encap, mux, bit, man]
                (EncapDataLossEnabled, BitDataLossEnabled)
          |||
          (
               hide int in
               (
               Encap  [encap, int]
                  (EncapVCDUID, EncapPCID)
               |[int]|
               DeEncap [encap, int]
                  (EncapDataLossEnabled, EncapVCDUID, EncapPCID,
                   ESDUNotLost)
               )
          )
     )
     []
     man ? ServiceIndicator : ServiceType
         ? MuxVCDUID : VCDUID
         ? MaxMSDUSize : Nat
           [ServiceIndicator Eq Mux] ;
     (
          VCLCServices [encap, mux, bit, man]
                (EncapDataLossEnabled, BitDataLossEnabled)
          |||
          (
               hide int in
               (
               Mux  [mux, int]
                  (MuxVCDUID)
               |[int]|
               DeMux [mux, int]
                  (MuxVCDUID)
               )
          )
     )
     []
```

```
     man ? ServiceIndicator : ServiceType
         ? BitVCDUID : VCDUID
           [ServiceIndicator eq Bit];
     (
          VCLCServices [encap, mux, bit, man]
                (EncapDataLossEnabled, BitDataLossEnabled)
          |||
          (
               hide int in
               (
               Bit  [bit, int]
                  (BitVCDUID)
               |[int]|
               DeBit [bit, int]
                  (BitVCDUID, BitDataLossEnabled, BitstreamDataNotLost)
               )
          )
     )
)

where
```

```
process Encap [encap, int]
          (EncapVCDUID : VCDUID,
           EncapPCID : APID) : noexit :=

(
     encap ! EncapVCDUID
           ! EncapPCID
           ? ESDU : OctetString ;

     int ! EncapVCDUID
         ! EncapPCID
         ! ESDU ;

     ENCAP [encap, int]
          (EncapVCDUID,
           EncapPCID)

)

endproc Encap
```

```
process DeEncap [encap, int]
          (EncapDataLossEnabled : Bool,
           EncapVCDUID : VCDUID,
           EncapPCID : APID,
           EncapDataLossFlag : ESDULossFlag) : noexit :=
(
     int ! EncapVCDUID
         ! EncapPCID
         ? ESDU : OctetString ;
     (
     [EncapDataLossEnabled eq false] ->
          (
               (
               encap ! EncapVCDUID
                     ! EncapPCID
                     ! ESDU ;

               DeEncap [encap, int]
                       (EncapDataLossEnabled,
                        EncapVCDUID,
                        EncapPCID,
                        EncapDataLossFlag)
               )
               []
               (
               DeEncap [encap, int]
                       (EncapDataLossEnabled,
                        EncapVCDUID,
                        EncapPCID,
                        EncapDataLossFlag)
               )
          )
     []
     [EncapDataLossEnabled eq true] ->
          (
               (
               encap ! ESDU
                     ! EncapVCDUID
                     ! EncapPCID
                     ! EncapDataLossFlag;

               DeEncap [encap, int]
                       (EncapDataLossEnabled,
                        EncapVCDUID,
                        EncapPCID,
                        ESDUNotLost)
               )
               []
               (
               DeEncap [encap, int]
                       (EncapDataLossEnabled,
                        EncapVCDUID,
                        EncapPCID,
                        ESDULost)
               )
          )
     )
)
endproc DeEncap
```

```
process Mux [mux, int]
            (MuxVCDUID : VCDUID) : noexit :=
(
     mux ! MuxVCDUID
         ? MuxPCID : APID
         ? MSDU : CCSDSPacket ;

     int ! MuxVCDUID
         ! MuxPCID
         ! MSDU ;

     MUX [mux, int]
         (MuxVCDUID)
)
endproc  Mux


process DeMux [mux, int]
              (MuxVCDUID : VCDUID) : noexit :=
(
     int ! MuxVCDUID
         ? MuxPCID : APID
         ? MSDU : CCSDSPacket ;
     (
          (
          mux ! MuxVCDUID
              ! MuxPCID
              ! MSDU ;

          DEMUX [mux, int]
                (MuxVCDUID)
          )
          []
          (
            DeMux [mux, int]
                  (MuxVCDUID)
          )
     )
)

endproc DeMux
```

```
process Bit [bit, int]
            (BitVCDUID : VCDUID) : noexit :=
(
     bit ! BitVCDUID
         ? BSDU : Bit ;

     int ! BitVCDUID
         ! BSDU ;

     BIT [bit, int]
         (BitVCDUID)

)

endproc Bit


process DeBit [bit, int]
              (BitVCDUID : VCDUID,
               BitDataLossEnabled: Bool,
               BitstreamDataLossFlag : BitstreamDataLossFlag): noexit :=
(
     int ! BitVCDUID
         ? BSDU : Bit ;
     (
     [BitDataLossEnabled eq false] ->
          (
               (
               bit ! BitVCDUID
                   ! BSDU ;

               DeBit [bit, int]
                     (BitVCDUID,
                      BitDataLossEnabled,
                      BitstreamDataLossFlag)
               )
               []
               (
               DeBit [bit, int]
                     (BitVCDUID,
                      BitDataLossEnabled,
                      BitstreamDataLossFlag)

               )
          )
     []
```

```
     [BitDataLossEnabled eq true] ->
          (
               (
               bit ! BitVCDUID
                   ! BitstreamDataLossFlag
                   ! BSDU ;

               DeBit [bit, int]
                     (BitVCDUID,
                      BitDataLossEnabled,
                      BitstreamDataNotLost)
               )
               []
               (
               DeBit [bit, int]
                     (BitVCDUID,
                      BitDataLossEnabled,
                      BitstreamDataLost)
               )
          )
     )
)

endproc DeBit

endproc VCLCServices

endspec
```

ANNEX A

VCLC PROTOCOL AND SERVICE TESTS

(THIS ANNEX **IS NOT** PART OF THE RECOMMENDATION)

Purpose:

This Annex details procedures for testing the VCLC protocol and service.

Virtual Channel Link Control Protocol Test 1

Name: Multiplexing.

Description: For each Virtual Channel the Multiplexing Layer constructs M_PDUs by multiplexing together M_SDUs and E_PDUs (both Version-1 CCSDS Packets) [5.3.8.1.2.1][1].

Inputs: A string of M_UNITDATA.requests and E_PDUs using the same VCDU-ID, but with different PCIDs (APIDs).

Configuration: Management Information concerning M_PDU size, such that, in conjunction with the above input, several M_PDUs are generated.

Expected Results: Several VCA_UNITDATA.requests, all on the same VCDU-ID, containing CCSDS Packets on the above different PCIDs.

---

[1]References in square brackets are to sections in the AOS Blue Book (reference [2]).

```
process vclcpt1 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Mux
         ! Succ(Succ(Succ(Succ(8))))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;   (* MaxMSDUSize *)

     man ! Encap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0)
         ! 8*8 ; (* MaxESDUSize *)
     (
     (
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,
                                             PacketSequenceCount(0,0,0,0,
                                                                 0,0,0,0,
                                                                 0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,0,0),
                             AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;
```

```
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,0,1)),
                               MakePacketSC(PacketSequenceUnSeg,
                                             PacketSequenceCount(0,0,0,0,
                                                                 0,0,0,0,
                                                                 0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,1,0),
                             AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,1,0,0),
              AddFront(Octet(0,0,0,0,0,1,0,1), NullOS)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;
     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt1
```

Virtual Channel Link Control Protocol Test 2

Name: Demultiplexing.

Description: The Multiplexing Layer extracts E_PDUs and M_SDUs from M_PDUs received from the VCA sublayer [5.3.8.1.2.2].

Inputs: A string of VCA_UNITDATA.indications on the same VCDU-ID as that in the Multiplexing test, containing the VCA_SDUs generated in the Multiplexing test.

Configuration: No specific configuration data is required.

Expected Results: A string of M_UNITDATA.indications and E_PDUs should be passed to the Multiplexing service user and de-Encapsulation Procedures in the reverse of the input to the Multiplexing test.

```
process vclcpt2 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
          (False,
           bDLF)
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (False,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)

)

where

process test [encap, mux, bit, vca, relmux, relbit, man]
             (eDLF : Bool,
              bDLF : Bool) : exit :=

hide success, failure in
(

     man ! DeMux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     man ! DeEncap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0) ;
     (
     (

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 1, 0, 0, 0, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                            addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 1, 0, 0, 0, 0, 1),
                                addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                  nullos)))))))))))))))
         ! VCDUNotLost ;
```

```
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? PCID : APID
         ? MSDU : CCSDSPacket ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (MakeMPDU(MakeMPDUHeader(MakeMPDUSpare(0,0,0,0,0),
                                     MakeMPDUFHP(0,0,0,0,0,0,0,0,1,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                        addfront(octet(0, 0, 0, 0, 0, 0, 1, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                          addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                           addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                            addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                                addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                                  nullos)))))))))))))
         ! VCDUNotLost ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? PCID : APID
         ? MSDU : CCSDSPacket ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString ;
     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt2
```

Virtual Channel Link Control Protocol Test 3

Name: Correct Function of First Header Pointer.

Description: The First Header Pointer in the M_PDU facilitates delimiting of the variable length SDUs contained within the M_PDU Packet Zone [5.3.8.2.2.2].

Inputs: Inputs should be provided to test the following conditions:

1) M_UNITDATA.requests such that a Packet Header is split between M_PDUs A and B, where M_PDU B contains another complete Packet Header.

2) M_UNITDATA.requests such that an M_PDU Packet Zone is produced containing only Packet data, i.e., no Header.

3) No M_PDU requests, but a VC_PDU Release Parameter event such that an M_PDU is created containing only a Fill Packet.

Configuration: Configuration data should be produced in conjunction with the described inputs above (the only important piece of Management Information is the VCDU/CVCDU Data Zone length).

Expected Results: In case 1), M_PDU A should have a First Header Pointer pointing to the first Header in the Packet Zone, and M_PDU B should have an FHP pointing to the complete Packet Header (ignoring the fractional continuation from M_PDU A). In case 2), the M_PDU FHP should be set to all ones. In case 3), the M_PDU FHP should be set to all ones minus one.

```
process vclcpt3 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
(
     hide encap, mux, bit, vca, relmux, relbit, man in
     (
          test [encap, mux, bit, vca, relmux, relbit, man]
          |[encap, mux, bit, vca, relmux, relbit, man]|
          VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                                (eDLF,
                                 bDLF,
                                 packetType,
                                 muxFill,
                                 bitFill)
     )
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(
     (

     man ! Mux
         ! Succ(Succ(Succ(Succ(8))))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;  (* MaxMSDUSize *)

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,

PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,0,0),
                             AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,

PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,1,0),
                             AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString
         [GetFHP(GetMPDUHeader(ConvertOSToMPDU(MPDU))) Eq
          MakeMPDUFHP(0,0,0,0,0,0,0,0,0,0,0)] ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,

PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1)),
                             AddFront(Octet(0,0,0,0,0,1,0,0),
                              AddFront(Octet(0,0,0,0,0,1,0,1),
                               AddFront(Octet(0,0,0,0,0,1,1,0),
                                AddFront(Octet(0,0,0,0,0,1,1,1),
                                 AddFront(Octet(0,0,0,0,1,0,0,0),
                                  AddFront(Octet(0,0,0,0,1,0,0,1),
                                   AddFront(Octet(0,0,0,0,1,0,1,0),
                                    AddFront(Octet(0,0,0,0,1,0,1,1),
                                     AddFront(Octet(0,0,0,0,1,1,0,0),
                                      AddFront(Octet(0,0,0,0,1,1,0,1),
                                       AddFront(Octet(0,0,0,0,1,1,1,0),
                                        AddFront(Octet(0,0,0,0,1,1,1,1),
                                         AddFront(Octet(0,0,0,1,0,0,0,0),
                                          AddFront(Octet(0,0,0,1,0,0,0,1),
                                           NullOS))))))))))))))) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString
         [GetFHP(GetMPDUHeader(ConvertOSToMPDU(MPDU))) Eq
          MakeMPDUFHP(0,0,0,0,0,0,0,0,1,0,0)] ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString
         [GetFHP(GetMPDUHeader(ConvertOSToMPDU(MPDU))) Eq NoPacketHeaderFHP] ;

     relmux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1),
                         MakeVCID(0,0,0,0,1,1)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString
         [GetFHP(GetMPDUHeader(ConvertOSToMPDU(MPDU))) Eq FillFHP] ;

     success ; exit
     )
     []
     (
          failure ; exit
     )
)
endproc test

endproc vclcpt3
```

Virtual Channel Link Control Protocol Test 4

Name: Correct function of the VC_PDU release parameter.

Description: Each VCDU-ID has associated with it parameters which define when the VC_PDU must be released for transmission [5.5.1.2].

Inputs: M_UNITDATA.requests on VCDU-ID A, constituting input to an M_PDU of length X, followed by a VC_PDU Release Parameter event.

Configuration: Management information for VCDU-ID A, such that the required M_PDU length is much greater than X, and that a VC_PDU Release Parameter event will occur.

Expected Results: A VCA_UNITDATA.request should be generated with a filled M_PDU. The Packet containing the fill data should have a header as follows:

```
Version:                000
Type:                   0 or 1
Secondary Header Flag:  0.
APID:                   11111111111
Sequence Flags:         11
Sequence Count:         00000000000000
Packet Length:          Number of fill octets - 1
```

```
process vclcpt4 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(
     (

     man ! Mux
         ! 8*8  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;  (* MaxMSDUSize *)

     man ! Encap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0)
         ! 8*8 ;  (* MaxESDUSize *)

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,
                               PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                             AddFront(Octet(0,0,0,0,0,0,0,0),
                              AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;
```

```
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,0,1)),
                               MakePacketSC(PacketSequenceUnSeg,
                               PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,1,0),
                             AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;

     relmux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1),
                         MakeVCID(0,0,0,0,1,1)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;

     success ; exit
     )
     []
     (
          failure ; exit
     )
)
endproc test

endproc vclcpt4
```

Virtual Channel Link Control Protocol Test 5

Name: VCA_SDU Length Check.

Description: The Channel Access Data Units (CADUs) for a Physical Channel must be of a fixed length over the lifetime of that channel [5.4.10.1]. The Multiplexing Layer must therefore produce VCA_SDUs which exactly fit the data zone of the VCDU/CVCDU to be transmitted over the Physical Channel in question.

Inputs: A string of M_UNITDATA.requests such that their overall length exceeds a certain length X.

Configuration: Management information such that the required length of the VCDU/CVCDU data zone is Y, where Y<X.

Expected Results: At least one VCA_UNITDATA.request should be produced, with a VCA_SDU of length Y.

```
process vclcpt5 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)
where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(
     (

     man ! Mux
         ! Succ(Succ(8))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;  (* MaxMSDUSize *)

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,
                               PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,0,0),
                             AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(0,0,0,0,0,1,0,0,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,
                               PacketSequenceCount(0,0,0,0,0,0,0,0,0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                            AddFront(Octet(0,0,0,0,0,0,1,0),
                             AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString
         [LengthOf(GetMPDUPZ(COnvertOSToMPDU(MPDU))) Eq Succ(Succ(8))] ;
     success ; exit
     )
     []
     (
          failure ; exit
     )
)
endproc test

endproc vclcpt5
```

Virtual Channel Link Control Protocol Test 6

Name: Packet Channel ID Checks

Description: The Multiplexing Layer should check the APID (PCID) fields of Packets contained in the M_PDUs received from the VCA sublayer, generating M_UNITDATA.indications for those Packets with User APIDs, and passing Packets with APIDs reserved for the Encapsulation service to the de-Encapsulation procedures. Currently, APIDs 0-2031 are User APIDs, 2032-2045 are unassigned reserved APIDs, 2046 is the Encapsulation APID, and 2047 is reserved for Fill Packets.

Inputs: A VCA_UNITDATA.indication (or string thereof) containing an M_PDU (or string thereof). Four complete Packets should be contained in the M_PDU(s), the first with a User APID, the second with the Encapsulation service APID, the third with an unassigned reserved APID, and the last with a Fill Packet APID.

Configuration: No specific configuration is required.

Expected Results: The first Packet should result in an M_UNITDATA.indication, the second should be passed to the de-Encapsulation procedures, the third should result in an error condition, and the last should be discarded.

```
process vclcpt6 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (False,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)

)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(
     (

     man ! DeMux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     man ! DeEncap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 1, 0, 0, 0, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                            addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 1, 0),
                               addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                                addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                                  addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                                   addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                                    nullos))))))))))))))))
         ! VCDUNotLost ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? PCID : APID
         ? MSDU : CCSDSPacket ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (MakeMPDU(MakeMPDUHeader(MakeMPDUSpare(0,0,0,0,0),
                                     MakeMPDUFHP(0,0,0,0,0,0,0,1,0,0,0)),
                      addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                        addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                          addfront(octet(0, 0, 0, 0, 0, 1, 1, 0),
                           addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                            addfront(octet(0, 0, 0, 0, 1, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 1, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                               addfront(octet(1, 1, 1, 1, 1, 1, 0, 1),
                                addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                                  addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                   addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                                    nullos)))))))))))))))))
         ! VCDUNotLost ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (MakeMPDU(MakeMPDUHeader(MakeMPDUSpare(0,0,0,0,0),
                                     MakeMPDUFHP(0,0,0,0,0,0,0,0,1,0,0)),
                      addfront(octet(0, 0, 0, 0, 1, 0, 1, 0),
                       addfront(octet(0, 0, 0, 0, 1, 0, 1, 1),
                        addfront(octet(0, 0, 0, 0, 1, 1, 0, 0),
                         addfront(octet(0, 0, 0, 0, 1, 1, 0, 1),
                          addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                           addfront(octet(1, 1, 1, 1, 1, 1, 1, 1),
                            addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                                addfront(octet(0, 0, 0, 0, 1, 1, 1, 0),
                                 addfront(octet(0, 0, 0, 0, 1, 1, 1, 1),
                                  addfront(octet(0, 0, 0, 1, 0, 0, 0, 0),
                                   addfront(octet(0, 0, 0, 1, 0, 0, 0, 1),
                                    nullos)))))))))))))))))
         ! VCDUNotLost ;

     success ; exit
     )
     []
     (
          failure ; exit
     )
)
endproc test

endproc vclcpt6
```

Virtual Channel Link Control Protocol Test 7

Name: Non-Version-1 CCSDS Packet.

Description: The Multiplexing Layer Service Data Unit is a Version-1 CCSDS Packet [5.3.6.2.c]. Packets with the Version field set to any other value should not be accepted by the Multiplexing Layer. The Multiplexing Layer should check that Packets are Version-1 whether received from the 'upper layer' (i.e., the Path Layer), or the 'lower layer' (i.e., VCA sublayer). In the first case CCSDS Packets appear in the service primitives; in the second case the Packets are embedded in Multiplexing Layer Protocol Data Units.

Inputs: Two M_UNITDATA.requests should be generated; the first with a Non-Version-1 version field, the second with a Version-1 version field. These inputs should be followed by an M_PDU Release event. A VCA_UNITDATA.indication should be generated where the VCA_SDU is an M_PDU containing two short Packets, one of which is a Non-Version-1 CCSDS Packet.

Configuration: No specific configuration is required.

Expected Results: The M_PDU release event should cause a VCA_UNITDATA.request to be generated, where the VCA_SDU contains only the Version-1 CCSDS Packet. The VCA_UNITDATA.request should be followed by only one Multiplexing indication, with the Packet being the Version-1 CCSDS Packet.

```
process vclcpt7 (eDLF : Bool,
                 bDLF : Bool,
                 packetType : PacketType,
                 muxFill : OctetString,
                 bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)

)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Mux
         ! Succ(Succ(Succ(Succ(8))))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,1,0,0))
         ! 8*8 ;  (* MaxMSDUSize *)

     man ! DeMux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     (
     (
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 1, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 1, 0, 0, 0, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                            addfront(octet(1, 1, 1, 1, 1, 1, 1, 1),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                              addfront(octet(0, 0, 1, 0, 0, 0, 1, 0),
                               addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                  addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                   addfront(octet(1, 1, 1, 1, 1, 1, 1, 1),
                                    nullos)))))))))))))))
         ! VCDUNotLost ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? PCID : APID
         ? MSDU : CCSDSPacket [GetVersion(GetPrimaryHeader(MSDU)) Eq Version1] ;
```

```
          (
          (
          mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,1,0,0))
              ! APID(0,0,0,0,0,1,0,0,0,1,0)
              ! MakeCCSDSPacket(MakePrimaryHeader
                                  (MakePacketID(Version(0,0,1),
                                                PacketType(0),
                                                SHAbsent,
                                                APID(0,0,0,0,0,1,0,0,0,1,0)),
                                   MakePacketSC(PacketSequenceUnSeg,
                                                 PacketSequenceCount(0,0,0,0,
                                                                     0,0,0,0,

0,0,0,0,0,0)),

PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                                AddFront(Octet(0,0,0,0,0,0,0,0),
                                 AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;
          failure ; exit
          )
          []

          (
          mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,1,0,0))
              ! APID(0,0,0,0,0,1,0,0,0,1,0)
              ! MakeCCSDSPacket(MakePrimaryHeader
                                  (MakePacketID(Version1,
                                                PacketType(0),
                                                SHAbsent,
                                                APID(0,0,0,0,0,1,0,0,0,0,1)),
                                   MakePacketSC(PacketSequenceUnSeg,
                                                 PacketSequenceCount(0,0,0,0,
                                                                     0,0,0,0,

0,0,0,0,0,0)),

PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                                AddFront(Octet(0,0,0,0,0,0,1,0),
                                 AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;
          success ; exit
          )
          )
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt7
```

Virtual Channel Link Control Protocol Test 8

Name: Inappropriate Grade Of Service

Description: The Multiplexing Layer is not permitted to operate over Grade 3 Virtual Channels [5.1.4.e].

Inputs: A correctly formatted M_UNITDATA.request on Virtual Channel X. A correctly formatted VCA_UNITDATA.indication on Virtual Channel X.

Configuration: Virtual Channel X should be configured as a Grade-3 channel.

Expected Results: Both inputs should result in error conditions.

Note – This test should probably be applied to management rather than the VCLC Protocol; it is included for completeness.

Virtual Channel Link Control Protocol Test 9

Name: Inappropriate VCDU-ID.

Description: Each VCDU-ID is associated with an upper-layer service interface [5.5.1.1.c]. Use of the multiplexing service should not be allowed on a VCDU-ID which is associated with a different service.

Inputs: A correctly formatted M_UNITDATA.request on VCDU-ID X. A correctly formatted VCA_UNITDATA.indication on VCDU-ID X.

Configuration: VCDU-ID X should be associated with an upper-layer service other than the Multiplexing service.

Expected Results: Both inputs should result in error conditions.

Note – This test should probably be applied to management rather than the VCLC Protocol; it is included for completeness.

Virtual Channel Link Control Protocol Test 10

Name: Unknown VCDU-ID.

Description: A VCDU-ID forms part of the M_UNITDATA.request and VCA_UNITDATA.indication.

Inputs: A correctly formatted M_UNITDATA.request using VCDU-ID X. A correctly formatted VCA_UNITDATA.indication using VCDU-ID X.

Configuration: Any management information which excludes VCDU-ID X.

Expected Results: Both inputs should result in error conditions.

```
process vclcpt10 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)
where
process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=
hide success, failure in
(
     (
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                            (MakePacketID(Version1,
                                          PacketType(0),
                                          SHAbsent,
                                          APID(0,0,0,0,0,1,0,0,0,1,0)),
                             MakePacketSC(PacketSequenceUnSeg,
                                           PacketSequenceCount(0,0,0,0,
                                                               0,0,0,0,
                                                               0,0,0,0,0,0)),
                             PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                          AddFront(Octet(0,0,0,0,0,0,0,0),
                           AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;
     failure ; exit
     )
     []
     (
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 1, 0, 0, 0, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                            addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 1, 0, 0, 0, 0, 1),
                                addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                  nullos)))))))))))))
         ! VCDUNotLost ;
     failure ; exit
     )
     []
     (
          success ; exit
     )
)
endproc test
endproc vclcpt10
```

Virtual Channel Link Control Protocol Test 11

Name: Inappropriate PCID.

Description: The multiplexing Layer should only accept M_UNITDATA.requests on Packet Channels which are unreserved; i.e., those Packet Channels between 0 and 2031 inclusive. (Should the Multiplexing Layer permit the passage of Fill Packets? Bearing in mind that the remote Multiplexing Layer will just discard them, no.)

Inputs: M_UNITDATA.requests on reserved PCIDs.

Configuration: No specific configuration is required.

Expected Results: These inputs should result in error conditions.

```
process vclcpt11 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Mux
         ! Succ(Succ(Succ(Succ(8))))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;  (* MaxMSDUSize *)
     (
     (
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                              (MakePacketID(Version1,
                                            PacketType(0),
                                            SHAbsent,
                                            APID(1,1,1,1,1,1,1,1,0,1,0)),
                               MakePacketSC(PacketSequenceUnSeg,
                                             PacketSequenceCount(0,0,0,0,
                                                                 0,0,0,0,
                                                                 0,0,0,0,0,0)),
                               PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                             AddFront(Octet(0,0,0,0,0,0,0,0),
                              AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;

     failure ; exit
     )
     []
     (
          success ; exit
     )
     )
)
endproc test

endproc vclcpt11
```

Virtual Channel Link Control Protocol Test 12

Name: Incorrect Packet Length field or defective M_PDU Packet Zone.

Description: The Multiplexing Layer uses the FHP to facilitate delimiting of the variable-length SDUs contained in the Packet Zone [5.3.8.2.2.2].

Inputs: A VCA_UNITDATA.indication where the FHP and the Packet Length field in a CCSDS Packet disagree.

Configuration: No specific configuration is required.

Expected Results: An error condition should occur.

```
process vclcpt12 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)

)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! DeMux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     man ! DeEncap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0) ;
     (
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 1, 0, 0, 0, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                            addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 1, 0, 0, 0, 0, 1),
                                addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                  nullos)))))))))))))
         ! VCDUNotLost ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? PCID : APID
         ? MSDU : CCSDSPacket ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (MakeMPDU(MakeMPDUHeader(MakeMPDUSpare(0,0,0,0,0),
                                     MakeMPDUFHP(0,0,0,0,0,0,0,0,1,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                        addfront(octet(0, 0, 0, 0, 0, 0, 1, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                          addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                           addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                            addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                                addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                                  nullos)))))))))))))
         ! VCDUNotLost ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString ;

          success ; exit
     )
)
endproc test

endproc vclcpt12
```

Virtual Channel Link Control Protocol Test 13

Name: Encapsulation.

Description: The CCSDS Packet produced by the Encapsulation Layer should have a certain Header format [5.3.8.2.1].

Inputs: Five correctly formatted E_UNITDATA.requests, with known E_SDU content.

Configuration: No specific configuration is required.

Expected Results: The resulting M_SDU should have the following format:

```
Version:                000
Type:                   as per set up information
Secondary Header Flag:  0
APID:                   11111111110
Sequence Flags:         11
Sequence Count:         see below
Packet Length:          Number of Octets in E_SDU - 1
User Data:              E_SDU
```

The five Sequence Counts should be 0, 1, 2, 3, 4.

```
process vclcpt13 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Mux
         ! Succ(Succ(8))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;  (* MaxMSDUSize *)

     man ! Encap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0)
         ! 8*8 ;  (* MaxESDUSize *)
     (
     (
     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,0,0,0),
              AddFront(Octet(0,0,0,0,0,0,0,1), NullOS)) ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,0,1,0),
              AddFront(Octet(0,0,0,0,0,0,1,1), NullOS)) ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;

(* Check contents of MPDU *)

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,1,0,0),
              AddFront(Octet(0,0,0,0,0,1,0,1), NullOS)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;

(* Check contents of MPDU *)

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,1,1,0),
              AddFront(Octet(0,0,0,0,0,1,1,1), NullOS)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;

(* Check contents of MPDU *)

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,1,0,0,0),
              AddFront(Octet(0,0,0,0,1,0,0,1), NullOS)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? MPDU : OctetString ;

(* Check contents of MPDU *)

     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt13
```

Virtual Channel Link Control Protocol Test 14

Name: De-Encapsulation

Description: The Encapsulation Layer receives E_PDUs (Version-1 CCSDS Packets) from the Multiplexing Layer, removes the E_SDU from the E_PDU and sends it to the Encapsulation Layer user with an E_UNITDATA.indication.

Inputs: An E_PDU from the Multiplexing Layer with known data content.

Configuration: No specific configuration is required.

Expected Results: An E_UNITDATA.indication with the E_SDU being identical to the E_PDU user data.

```
process vclcpt14 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (False,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)

)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! DeMux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     man ! DeEncap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0) ;
     (
     (
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                       addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                            addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                               addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                                addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                                  nullos)))))))))))))
         ! VCDUNotLost ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
             [ESDU Eq addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                        nullos))] ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (MakeMPDU(MakeMPDUHeader(MakeMPDUSpare(0,0,0,0,0),
                                     MakeMPDUFHP(0,0,0,0,0,0,0,0,1,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                        addfront(octet(0, 0, 0, 0, 0, 0, 1, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                          addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                           addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                            addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                                addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                                  nullos)))))))))))))
         ! VCDUNotLost;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
             [ESDU Eq addfront(octet(0, 0, 0, 0, 0, 0, 1, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                        nullos))] ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
             [ESDU Eq addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                        nullos))] ;

     success ; exit

     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt14
```

Virtual Channel Link Control Protocol Test 15

Name: Inappropriate Grade Of Service

Description: The Encapsulation Layer is not permitted to operate over Grade-3 Virtual Channels [5.1.4.e].

Inputs: A correctly formatted E_UNITDATA.request on Virtual Channel X.

Configuration: Virtual Channel X should be configured as a Grade-3 channel.

Expected Results: This input should result in an error condition.

Note – This test should probably be applied to management rather than the VCLC Protocol; it is included for completeness.

Virtual Channel Link Control Protocol Test 16

Name: Inappropriate VCDU-ID.

Description: Each VCDU-ID is associated with an upper-layer service interface [5.5.1.1.c]. Use of the Encapsulation service should not be allowed on a VCDU-ID which is associated with a different service.

Inputs: A correctly formatted E_UNITDATA.request on VCDU-ID X.

Configuration: VCDU-ID X should be associated with an upper-layer service other than the Encapsulation service.

Expected Results: This input should result in an error condition.

Note – This test should probably be applied to management rather than the VCLC Protocol; it is included for completeness.

Virtual Channel Link Control Protocol Test 17

Name: Unknown VCDU-ID.

Description: A VCDU-ID forms part of the E_UNITDATA.request.

Inputs: A correctly formatted E_UNITDATA.request using VCDU-ID X.

Configuration: Any Management Information which excludes VCDU-ID X.

Expected Results: This input should result in an error condition.

```
process vclcpt17 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     (
     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1),
                        MakeVCID(0,0,0,0,1,1))
           ! APID(0,0,0,0,0,1,0,0,0,1,0)
           ! AddFront(Octet(0,0,0,0,0,0,0,1), NullOS) ;
     failure ; exit
     )
     []
     (
          success ; exit
     )
)
endproc test

endproc vclcpt17
```

Virtual Channel Link Control Protocol Test 18

Name: Inappropriate PCID.

Description: Currently, only one Packet Channel Identifier (PCID) is reserved for the Encapsulation Service: 2046.

Inputs: E_UNITDATA.requests on non-Encapsulation PCIDs.

Configuration: No specific configuration is required.

Expected Results: These inputs should result in error conditions.

```
process vclcpt18 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Mux
         ! Succ(Succ(Succ(Succ(8))))  (* PZSize *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;   (* MaxMSDUSize *)

     man ! Encap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,1,1,1,1,1,1,1,1,0)
         ! 8*8 ;   (* MaxESDUSize *)
     (
     (
     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(0,0,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,0,0,0),
              AddFront(Octet(0,0,0,0,0,0,0,1), NullOS)) ;

     failure ; exit
     )
     []
     (
          success ; exit
     )
     )
)
endproc test

endproc vclcpt18
```

Virtual Channel Link Control Protocol Test 19

Name: E_PDU Discontinuities

Description: The Encapsulation Protocol should, if required, generate an E_SDU Loss Flag derived from the Sequence Count field of the E_PDU.

Inputs: A sequence of three E_PDUs, with the Sequence Count fields set to 32, 34 and 35.

Configuration: No specific configuration is required.

Expected Results: Three E_UNITDATA.indications should be generated, the first with a Data Loss Flag set to false, the second with a Data Loss Flag set to True, and the third with a Data Loss flag set to false.

```
process vclcpt19 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (true,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)

)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! DeMux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     man ! DeEncap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0) ;
     (

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (makempdu(makempduheader(makempduspare(0,0,0,0,0),
                                     makempdufhp(0,0,0,0,0,0,0,0,0,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                       addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                        addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                          addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                           addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                            addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                              addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                               addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                                addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                                  nullos)))))))))))))))
         ! VCDUNotLost ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
           ! ESDUNotLost ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertMPDUToOS
            (MakeMPDU(MakeMPDUHeader(MakeMPDUSpare(0,0,0,0,0),
                                     MakeMPDUFHP(0,0,0,0,0,0,0,0,1,0,0)),
                      addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                       addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                        addfront(octet(0, 0, 0, 0, 0, 0, 1, 0),
                         addfront(octet(0, 0, 0, 0, 0, 0, 1, 1),
                          addfront(octet(0, 0, 0, 0, 0, 1, 1, 1),
                           addfront(octet(1, 1, 1, 1, 1, 1, 1, 0),
                            addfront(octet(1, 1, 0, 0, 0, 0, 0, 0),
                             addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                              addfront(octet(0, 0, 0, 0, 0, 0, 0, 0),
                               addfront(octet(0, 0, 0, 0, 0, 0, 0, 1),
                                addfront(octet(0, 0, 0, 0, 0, 1, 0, 0),
                                 addfront(octet(0, 0, 0, 0, 0, 1, 0, 1),
                                  nullos)))))))))))))
         ! VCDUNotLost ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
           ! ESDULost ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
           ! ESDUNotLost ;
     success ;
     exit
     )
     []
     (
          failure ; exit
     )
)
endproc test

endproc vclcpt19
```

Virtual Channel Link Control Protocol Test 20

Name: B_PDU Construction.

Description: The B_PDU has a certain format [5.3.8.2.3].

Inputs: Eight correctly formatted BITSTREAM.requests, with known bit contents. (NOTE: hippo cannot properly identify a plain '0' as a bit, so the special values Bit0 and Bit1 are being used.)

Configuration: The B_PDU Construction procedure should be configured to have a Bitstream Data Zone of eight bits, to produce a B_PDU.

Expected Results: Each resulting B_PDU should have the following format:

```
Spare:                        00
Bitstream Data Pointer:       11111111111111
Bitstream Data Zone:          01110001 (from input bits)
```

Note – We are currently constraining the Bitstream Data Zone to be an integral number of octets. The AOS Blue Book (reference [2]) does not specify that this must be so, but as the VCDU Data Unit Zone must be an integral number of octets [5.4.9.2.1.3.a], the VCA_SDU must also. This forces the B_PDU to be an integral number of octets, forcing the Bitstream Data Zone to be so too.

```
process vclcpt20 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Bit                  (* Start Bitstream *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8 ;                  (* BDZ Size *)

     (
     (
     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit0 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit0 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit0 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit0 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;
```

```
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BPDU : OctetString
           [BPDU Eq
            AddFront(Octet(0,0,1,1,1,1,1,1),
             AddFront(Octet(1,1,1,1,1,1,1,1),
              AddFront(Octet(0,1,1,1,0,0,0,1), NullOS)))]   ;
              (* Spare is "00"
                 BDP is "11111111111111" to indicate all bits are user data
                 BDZ is "01110001"--"01110001" is user data
              *)


     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt20
```

Virtual Channel Link Control Protocol Test 21

Name: Bitstream Extraction.

Description: The Bitstream procedure receives a B_PDU from the VCA layer, extracts the bits from the Bitstream Data Zone in the B_PDU, and delivers them to the Bitstream user.

Inputs: A B_PDU from the VCA containing known data.

Configuration: No special configuration required.

Expected Results: The eight B_SDUs should have the following format:

BSDU1:	0
BSDU2:	1
BSDU3:	1
BSDU4:	1
BSDU5:	0
BSDU6:	0
BSDU7:	0
BSDU8:	1

```
process vclcpt21 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    false,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! DeBit        (* Start Bitstream Extraction *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     (
     (
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertBPDUToOS
            (MakeBPDU(MakeBPDUHeader(BPDUSpare,
                                     NoFillDataBDP),
                      AddLSB(1,
                       AddLSB(0,
                        AddLSB(0,
                         AddLSB(0,
                          AddLSB(1,
                           AddLSB(1,
                            AddLSB(1,
                             AddLSB(0, NullBS)))))))))
         ! VCDUNotLost ;
```

```
     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU1 : Bit
         [BSDU1 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU2 : Bit
         [BSDU2 Eq 1] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU3 : Bit
         [BSDU3 Eq 1] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU4 : Bit
         [BSDU4 Eq 1] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU5 : Bit
         [BSDU5 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU6 : Bit
         [BSDU6 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU7 : Bit
         [BSDU7 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU8 : Bit
         [BSDU8 Eq 1] ;


     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt21
```

Virtual Channel Link Control Protocol Test 22

Name: B_PDU Construction with Release.

Description: The B_PDU Bitstream Data Zone has fill data added in order to send out a complete B_PDU.

Inputs: Four correctly formatted BITSTREAM.requests, with known bit contents. (NOTE: hippo cannot properly identify a plain '0' as a bit, so the special values Bit0 and Bit1 are being used.)

Configuration: The B_PDU Construction procedure should be configured to have a Bitstream Data Zone of eight bits, to produce a B_PDU.

Expected Results: Each resulting B_PDU should have the following format:

```
Spare:                  00
Bitstream Data Pointer: 11111111111111
Bitstream Data Zone:    01110101 (0,1,1,1 input 0101 is fill)
```

Note – We are currently constraining the Bitstream Data Zone to be an integral number of octets. The AOS Blue Book (reference [2]) does not specify that this must be so, but as the VCDU Data Unit Zone must be an integral number of octets [5.4.9.2.1.3.a], the VCA_SDU must also. This forces the B_PDU to be an integral number of octets, forcing the Bitstream Data Zone to be so too.

```
process vclcpt22 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    AddMSB(0, AddMSB(1, NullBS)))
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Bit                   (* Start Bitstream *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8 ;                   (* BDZ Size *)

     (
     (
     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit0 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! Bit1 ;

     relbit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BPDU : OctetString
           [BPDU Eq
            AddFront(Octet(0,0,0,0,0,0,0,0),
             AddFront(Octet(0,0,0,0,0,0,1,1),
              AddFront(Octet(0,1,1,1,0,1,0,1), NullOS)))]   ;
              (* Spare is "00"
                 BDP is "00000000000011" to indicate four bits of user data
                  (BDP points to number of bit-1 [5.3.8.2.3.2.b]
                 BDZ is "01110101"--"0111" is user data, "0101" is fill
              *)

     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt22
```

Virtual Channel Link Control Protocol Test 23

Name: B_PDU Construction with Release.

Description: The B_PDU Bitstream Data Zone has only fill data to send out a complete B_PDU.

Inputs: A release signal without any user data.

Configuration: The B_PDU Construction procedure should be configured to have a Bitstream Data Zone of eight bits, to produce a B_PDU.

Expected Results: Each resulting B_PDU should have the following format:

```
Spare:                        00
Bitstream Data Pointer:       11111111111111
Bitstream Data Zone:          01010101 (01010101 is fill)
```

```
process vclcpt23 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    bDLF,
                    packetType,
                    muxFill,
                    AddMSB(0, AddMSB(1, NullBS)))
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! Bit                  (* Start Bitstream *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8 ;                  (* BDZ Size *)

     (
     (
     relbit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BPDU : OctetString
           [BPDU Eq
            AddFront(Octet(0,0,1,1,1,1,1,1),
             AddFront(Octet(1,1,1,1,1,1,1,0),
              AddFront(Octet(0,1,0,1,0,1,0,1), NullOS)))]   ;
              (* Spare is "00"
                 BDP is "11111111111110" to indicate no valid user data
                 BDZ is "01010101"--"01010101" is fill
              *)

     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt23
```

Virtual Channel Link Control Protocol Test 24

Name: Bitstream Extraction with Data Loss.

Description: The Bitstream procedure receives a B_PDU from the VCA layer, extracts the bits from the Bitstream Data Zone in the B_PDU, and delivers them to the Bitstream user. The VCA_UNITDATA.indication is accompanied by a VCDU Loss Flag.

Inputs: A B_PDU from the VCA containing known data.

Configuration: No special configuration required.

Expected Results: The eight B_SDUs should have the following format:

```
BSDU1:                    0
BSDU2:                    1
BSDU3:                    1
BSDU4:                    1
BSDU5:                    0
BSDU6:                    0
BSDU7:                    0
BSDU8:                    1
```

```
process vclcpt24 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    false,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! DeBit        (* Start Bitstream Extraction *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     (
     (
     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertBPDUToOS
            (MakeBPDU(MakeBPDUHeader(BPDUSpare,
                                     NoFillDataBDP),
                      AddLSB(1,
                       AddLSB(0,
                        AddLSB(0,
                         AddLSB(0,
                          AddLSB(1,
                           AddLSB(1,
                            AddLSB(1,
                             AddLSB(0, NullBS)))))))))
         ! VCDUNotLost ;
```

```
     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU1 : Bit [BSDU1 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU2 : Bit [BSDU2 Eq 1] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU3 : Bit [BSDU3 Eq 1] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU4 : Bit [BSDU4 Eq 1] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU5 : Bit [BSDU5 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU6 : Bit [BSDU6 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU7 : Bit [BSDU7 Eq 0] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU8 : Bit [BSDU8 Eq 1] ;


     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt24
```

Virtual Channel Link Control Protocol Test 25

Name: Bitstream Extraction with Data Loss and Data Lost.

Description: The Bitstream procedure receives a B_PDU from the VCA layer, extracts the bits from the Bitstream Data Zone in the B_PDU, and delivers them to the Bitstream user. The VCA_UNITDATA.indication is accompanied by a VCDU Loss Flag.

Inputs: A B_PDU from the VCA containing known data. The first will indicate data was lost.

Configuration: The Bitstream Extraction should expect a Data Loss flag from the VCA and provide one to the user.

Expected Results: The eight B_SDUs should have the following format:

```
BSDU9:                      1, BitstreamDataLost
BSDU10:                     0, BitstreamDataNotLost
BSDU11:                     0, BitstreamDataNotLost
BSDU12:                     0, BitstreamDataNotLost
BSDU13:                     1, BitstreamDataNotLost
BSDU14:                     1, BitstreamDataNotLost
BSDU15:                     1, BitstreamDataNotLost
BSDU16:                     0, BitstreamDataNotLost
```

```
process vclcpt25 (eDLF : Bool,
                  bDLF : Bool,
                  packetType : PacketType,
                  muxFill : OctetString,
                  bitFill : BitString) : noexit :=
hide encap, mux, bit, vca, relmux, relbit, man in
(
     test [encap, mux, bit, vca, relmux, relbit, man]
     |[encap, mux, bit, vca, relmux, relbit, man]|
     VCLCProtocols [encap, mux, bit, vca, relmux, relbit, man]
                   (eDLF,
                    True,
                    packetType,
                    muxFill,
                    bitFill)
)

where

process test [encap, mux, bit, vca, relmux, relbit, man] : exit :=

hide success, failure in
(

     man ! DeBit        (* Start Bitstream Extraction *)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1)) ;

     (
     (

     vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! ConvertBPDUToOS
            (MakeBPDU(MakeBPDUHeader(BPDUSpare,
                                     NoFillDataBDP),
                      AddLSB(0,
                       AddLSB(1,
                        AddLSB(1,
                         AddLSB(1,
                          AddLSB(0,
                           AddLSB(0,
                            AddLSB(0,
                             AddLSB(1, NullBS)))))))))
         ! VCDULost ;
```

```
     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU9 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU9 Eq 1) and (BDLF Eq BitstreamDataLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU10 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU10 Eq 0) and (BDLF Eq BitstreamDataNotLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU11 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU11 Eq 0) and (BDLF Eq BitstreamDataNotLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU12 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU12 Eq 0) and (BDLF Eq BitstreamDataNotLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU13 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU13 Eq 1) and (BDLF Eq BitstreamDataNotLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU14 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU14 Eq 1) and (BDLF Eq BitstreamDataNotLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU15 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU15 Eq 1) and (BDLF Eq BitstreamDataNotLost)] ;

     bit ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ? BSDU16 : Bit
         ? BDLF : BitstreamDataLossFlag
         [(BSDU16 Eq 0) and (BDLF Eq BitstreamDataNotLost)] ;


     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcpt25
```

Virtual Channel Link Control Service Test 1

Name: Data Transfer

Description: The Multiplexing Service provides transfer of Version-1 CCSDS Packets across the SLS. These Packets may be passed to the Multiplexing Layer by use of the M_UNITDATA.request, or they may be passed over an internal VCLC interface from the Encapsulation procedures. The Packets are multiplexed together for transmission over a particular VC, identified by a VCDU-ID.

Inputs: A string of M_UNITDATA.requests and E_PDUs, all on the same VCDU-ID, but with the M_UNITDATA.requests being on several different PCIDs, all having a known data content.

Configuration: No specific configuration is required.

Expected Results: A string of M_UNITDATA.indications and E_PDUs, on the same VCDU-ID and PCIDs, with the same known data content as the M_UNITDATA.requests.

```
process vclcst1 (eDLF : Bool,
                 bDLF : Bool) : noexit :=
hide encap, mux, bit, man in
(
     test [encap, mux, bit, man]
     |[encap, mux, bit, man]|
     VCLCServices [encap, mux, bit, man]
                  (False,
                   bDLF)
)

where

process test [encap, mux, bit, man] : exit :=

hide success, failure in
(

     man ! Mux
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! 8*8 ;   (* MaxMSDUSize *)

     man ! Encap
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(1,1,1,1,1,1,1,1,1,1,0)
         ! 8*8 ; (* MaxESDUSize *)
     (
     (
     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ! MakeCCSDSPacket(MakePrimaryHeader
                            (MakePacketID(Version1,
                                          PacketType(0),
                                          SHAbsent,
                                          APID(0,0,0,0,0,1,0,0,0,1,0)),
                             MakePacketSC(PacketSequenceUnSeg,
                                           PacketSequenceCount(0,0,0,0,
                                                               0,0,0,0,
                                                               0,0,0,0,0,0)),
                             PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                           AddFront(Octet(0,0,0,0,0,0,0,0),
                            AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))) ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,1,1,0,1,1)
         ! MakeCCSDSPacket(MakePrimaryHeader
                            (MakePacketID(Version1,
                                          PacketType(0),
                                          SHAbsent,
                                          APID(0,0,0,0,0,1,1,1,0,1,1)),
                             MakePacketSC(PacketSequenceUnSeg,
                                           PacketSequenceCount(0,0,0,0,
                                                               0,0,0,0,
                                                               0,0,0,0,0,0)),
                             PacketLength(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)),
                           AddFront(Octet(0,0,0,0,0,0,1,0),
                            AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;
```

```
     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ! AddFront(Octet(0,0,0,0,0,1,0,0),
              AddFront(Octet(0,0,0,0,0,1,0,1), NullOS)) ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,0,0,0,1,0)
         ? MSDU : CCSDSPacket
           [GetUserData(MSDU) Eq
            AddFront(Octet(0,0,0,0,0,0,0,0),
             AddFront(Octet(0,0,0,0,0,0,0,1), NullOS))] ;

     mux ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
         ! APID(0,0,0,0,0,1,1,1,0,1,1)
         ? MSDU : CCSDSPacket
           [GetUserData(MSDU) Eq
            AddFront(Octet(0,0,0,0,0,0,1,0),
             AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))] ;

     encap ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,0,1), MakeVCID(0,0,0,0,1,1))
           ! APID(1,1,1,1,1,1,1,1,1,1,0)
           ? ESDU : OctetString
             [ESDU Eq
              AddFront(Octet(0,0,0,0,0,1,0,0),
               AddFront(Octet(0,0,0,0,0,1,0,1), NullOS))] ;



     success ; exit
     )
     []
     (
          failure ; exit
     )
     )
)
endproc test

endproc vclcst1
```

Virtual Channel Link Control Service Test 2

Name: Data Transfer.

Description: The Encapsulation Service provides transfer of non-CCSDS structured delimited, octet aligned data units across the SLS.

Inputs: A string of E_UNITDATA.requests with known E_SDU content.

Configuration: No specific configuration is required.

Expected Outputs: A string of E_UNITDATA.indications with E_SDU content identical to the original E_UNITDATA.requests.

Virtual Channel Link Control Service Test 3

Name: Data Transfer

Description: The Bitstream Service is used to transfer Bitstream data across the SLS.

Inputs: A string of BITSTREAM.requests containing known data.

Configuration: No specific configuration is required.

Expected Results: A number of BITSTREAM.requests containing the same data as input, though not necessarily in the same groupings.