

Octet Sequences for Upper-Layer OSI  
to Support Basic Communications Applications

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This document states particular octet sequences that comprise the OSI upper-layer protocols (Session, Presentation and ACSE) when used to support applications with "basic communications requirements". These include OSI application protocols such as X.400 P7 and Directory Access Protocol, and "migrant" protocols, originally defined for use over other transports.

As well as the octet sequences which are the supporting layer headers (and trailers) around the application data, this document includes some tutorial material on the OSI upper layers.

An implementation that sends the octet sequences given here, and interprets the equivalent protocol received, will be able to interwork with an implementation based on the base standard, when both are being used to support an appropriate application protocol.

Table of Contents

|   |    |
|---|----|
| 1. Introduction .....   | 2  |
| 2. General .....  | 3  |
| 2.1 Subdivisions of "basic communication applications" .....  | 3  |
| 2.2 Conformance and interworking .....                        | 5  |
| 2.3 Relationship to other documents .....                     | 5  |
| 3. Contexts and titles .....                                  | 6  |
| 3.1 The concepts of abstract and transfer syntax .....        | 6  |
| 3.2 Use of presentation context by cookbook applications..... | 7  |
| 3.3 Processing Presentation-context-definition-list .....     | 8  |
| 3.4 Application context .....                                 | 9  |
| 3.5 APtitles and AEqualifiers .....                           | 9  |
| 4. What has to be sent and received .....                     | 10 |
| 4.1 Sequence of OSI protocol data units used .....            | 10 |
| 4.2 Which OSI fields are used .....                           | 12 |

|  |    |
|--|----|
| 4.3 Encoding methods and length fields .....             | 14 |
| 4.3.1 Session items .....                                | 14 |
| 4.3.2 ASN.1/BER items (Presentation and ACSE) .....      | 14 |
| 4.4 BER Encoding of values for primitive datatypes ..... | 15 |
| 4.5 Unnecessary constructed encodings .....              | 16 |
| 5. Notation .....  | 16 |
| 6. Octet sequences .....                                 | 17 |
| 6.1 Connection request message .....                     | 17 |
| 6.2 Successful reply to connection setup .....           | 20 |
| 6.3 Connection rejection .....                           | 22 |
| 6.4 Data-phase TSDU .....                                | 23 |
| 6.5 Closedown - release request .....                    | 24 |
| 6.6 Closedown - release response .....                   | 25 |
| 6.7 Deliberate abort .....                               | 25 |
| 6.8 Provider abort .....                                 | 27 |
| 6.9 Abort accept .....                                   | 27 |
| 7. References .....                                      | 27 |
| 8. Other notes .....                                     | 28 |
| 9. Security Considerations .....                         | 29 |
| 10. Author's Address .....                               | 29 |

## 1. Introduction

The upper-layer protocols of the OSI model are large and complex, mostly because the protocols they describe are rich in function and options. However, for support of most applications, only a limited portion of the function is needed. An implementation that is not intended to be a completely general platform does not need to implement all the features. Further, it need not reflect the structuring of the OSI specifications - the layer of the OSI model are purely abstract.

This document presents the protocol elements required by the OSI upper layers when supporting a connection-oriented application with only basic communication requirements - that is to create a connection, optionally negotiate the data representation, send/receive data, close a connection and abort a connection. Optionally, data may be sent on the connection establishment, closing and abort messages.

In this document, the protocol elements needed are given in terms of the octet sequences that comprise the 'envelope' around the application data. The envelope and its enclosing data form a Transport Service Data Unit (TSDU) that can be passed via the OSI Transport Service [ISO8072] (which in turn may be supported as specified in [RFC1006] or any class of the OSI Transport Protocol [ISO8073]).

The octet sequences to be sent and the description of the alternative forms that may be received are equivalent to an informal re-specification of the relevant parts of the upper-layer protocols. The "relevant parts" are determined by the requirements of the supported applications (this is a reflexive definition! - if application Z needs something that is not here, it is not supported). The formal specifications remain the base standards, the appropriate profiles and the requirements of the application. However, an implementation based on this document will be able to interwork with an implementation based directly on the full standards when both are supporting a basic communication application. The "full" implementation will exhibit only part of its potential behaviour, since the application will only invoke part.

In addition to the octet sequences, the document includes some tutorial material.

## 2. General

### 2.1 Subdivisions of "basic communication applications"

Distinctions can be made within the "basic communication applications", as defined above, based on how much use they make of the OSI upper-layer services, and thus how much of the protocol described in this memo will be used to support a particular application. One distinction is:

- a) whether application data is sent on the connection establishment, close and abort, or only during "data phase" on an established connection; OR
- b) whether the application data is of only one kind (abstract syntax) and one format (transfer syntax) or more than one (i.e., how much use is made of the Presentation layer syntax negotiation and identification features)

Further distinctions are possible, but in this memo, elements of protocol needed (or not needed) by four groups of "basic communications application" are identified. All groups have "basic communications requirements" in requiring only connection, data transfer and (perhaps) orderly release of connection. The four groups are:

Group I: applications which send data only on an established connection, and use a single abstract and transfer syntax.

Group II: applications which send data on connection establishment and release and use a single abstract and transfer syntax.

Group III: applications that send data of only one kind (one abstract syntax) on the connection, but which have more than one format (transfer syntax) specified (they use the Presentation context negotiation facility).

Group IV: applications that will send data of several kinds on the connection (and which must therefore distinguish on each write which kind is being sent).

Group III applications are equivalent to Group I (or possibly Group II) after the establishment exchange has negotiated the particular transfer syntax that will be used on the connection.

Possible examples of the Groups are:

Group I: Application protocols designed for use over transport-level protocols. Typically these are non-OSI protocols "migrated" to an OSI environment. X Window System protocol is an example.

Group II: OSI-originated protocols with simple requirements, including many of the ROSE-based ones, such as Directory Access Protocol.

Group III: Protocols that can be treated as Group I, but for which more than one encoding of the data is known, such as a standardised one and a system-specific one - all implementations understand the standard encoding, but Presentation layer negotiation allows like-implementations to use their internal encoding for transfer, without loss of general interworking. The same could apply to OSI protocols.

Group IV: OSI protocols with multiple abstract syntaxes (but with each individual message from a single abstract syntax) that do not use any of the special Session functional units - X.400 P7 is an example.

Some of the OSI protocols that are not included are those that use more than one abstract syntax in a single message (such as FTAM or Transaction Processing) or use Session functional units (RTSE-based protocols, Virtual Terminal).

## 2.2 Conformance and interworking

The protocol elements specified in this memo correspond to the kernel functional units of Session, Presentation and ACSE, and the duplex functional unit of Session.

The octet sequences given below are derived from the specifications in the International Standards for the protocols Session [ISO8327], Presentation [ISO8822] and ACSE [ISO8650]. The intention of this memo is to summarise those specifications, as applicable to the supported application groups, so that an implementation could be developed without direct reference to the original standards, but capable of interworking with implementations that had made direct reference. The OSI standards (especially Presentation) allow considerable flexibility in the encoding of the protocol data units. Accordingly, this memo defines particular octet sequences to be sent, and describes the variations that can be expected in data received from an implementation based directly on the OSI standards, rather than on this cookbook. It is intended that an implementation that sends these sequences and that is capable of interpreting the variations described will be fully able to interwork with an implementation based directly on the OSI standards. An implementation that is only capable of interpreting the octet sequences specified in this memo for transmission may not be able to interwork with standards-based implementations.

The intent is to be able to interwork with conformant implementations in support of the relevant application (or group of applications). Some of the OSI standards have conformance requirements that go beyond that necessary for successful interworking, including detection of invalid protocol. Tests for conformance sometimes go beyond the strict conformance requirements of the standard. Consequently an implementation based on this memo may or may not be able to formally claim conformance to the International Standard. It may be able to legitimately claim conformance, but fail a conformance test, if the test is over-specified. (Efforts are being made to correct this, but in the meantime, the target is interworking with conformant implementations.)

## 2.3 Relationship to other documents

The flexibility allowed in the Session, Presentation and ACSE standards is restricted in the Common Upper-Layer Requirements Part 1 [CULR-1]). This is a proposed International Standardised Profile (pdISP 11188-1) that can be assumed to be obeyed by most implementations. This memo applies the restrictions of CULR-1, especially where these concern maximum sizes of fields and the like. Points where advantage is taken of a CULR-1 limitation are

noted.

Additional parts of CULR are under development. Part 3 [CULR-3] covers the protocol elements needed for "basic communications applications", and is being developed in (informal) liaison with this memo. CULR-3 is presented as a normal profile, largely consisting of prescribed answers to the questions in the PICS (Protocol Implementation Conformance Statement) of the three protocols. CULR-3 does not make the distinction between the four Groups. An implementation of this memo (at least if it supported Group IV) would be able to claim conformance to CULR-3, with the possible exception of any more-than-interworking conformance requirements inherited by CULR-3 from the base standards.

An extension [XTI/mOSI] to the X/Open Transport Interface [XTI] is shortly to be published as a preliminary specification. This defines an API to the OSI upper-layers, again as appropriate to a basic communications application. XTI/mOSI would be usable as an interface to support applications in groups I, II and III, and possibly group IV.

### 3. Contexts and titles

#### 3.1 The concepts of abstract and transfer syntax

OSI includes the concepts of "abstract syntax" and "transfer syntax". These are terms for the content (abstract syntax) and format "on-the-line" (transfer syntax) of the protocol elements. The combination of an abstract syntax and transfer syntax is called a presentation context.

Application protocols devised explicitly under OSI auspices have used ASN.1 for the definition of the abstract syntax, and nearly all use the Basic Encoding Rules applied to the ASN.1 to define the transfer syntax. However, there is no such requirement in OSI in general or in OSI Presentation, and still less is there any requirement to change the representation of existing application protocols to ASN.1 (for their definition) or BER (for their transmission). It is not generally realised (even in OSI circles) that all communicating applications, in all environments, are using some form of these, although under different names and without the explicit identification that the OSI Presentation provides. OSI separates the identification of the content and format of the data from the addressing.

Formal specifications of non-OSI application protocols (such as TELNET, FTP, X Windows System) generally do not use ASN.1, but will invariably be found to define abstract and transfer syntaxes. For a

less formalised protocol used between similar systems, the abstract syntax may be defined simply in programming language structures, and the transfer syntax determined by how some compiler represents this in memory.

The OSI Presentation protocol requires that "names" be assigned to the abstract and transfer syntaxes of the application data that is carried. The names are always object identifiers ("oid"): globally unique names assigned hierarchically. Presentation supports the negotiation of a transfer syntax for a particular abstract syntax - several can be offered and one selected.

This transfer syntax negotiation facility may be especially useful for non-ASN.1 syntaxes where there is more than one representation available (perhaps differing in byte-ordering or character code). In such a case, on the connection establishment, all of the transfer syntaxes supported by the initiator are offered, and any one of these accepted by the responder, at its own choice. If the two systems share some "native" format they can negotiate that, avoiding transformation into and out of a more general format that is used for interworking with unlike systems. The same applies to an ASN.1-defined abstract syntax, but in practice non-BER encodings of ASN.1 are rare.

### 3.2 Use of presentation context by cookbook applications

An application protocol not originally specified with OSI Presentation in mind (a "migrant" protocol) will not normally need to identify the abstract and transfer syntaxes being used - they are known by some other means (effectively inferred from the addressing). A generic (anonymous, if you like) name for both syntaxes can be used and [CULR-3] defines object identifiers for "anonymous" abstract and transfer syntax names (currently called "default", but this is expected to change).

In some cases object identifier names will be assigned for the syntaxes of a migrant application protocol. If these exist, they should be used. However, since the processing required will be the same, it will be legitimate to offer both the generic and specific names, with the responder accepting the specific (if it knew it) and the generic if the specific were not known - this will provide a migration option if names are assigned to the syntaxes after implementations are deployed using the generic names.

For abstract syntaxes defined in ASN.1 object identifier names will have been assigned to the abstract syntax with the specification. This name **MUST** be used to identify the abstract syntax. The transfer syntax will most often be the Basic Encoding Rules (BER) object id,

but alternatives (e.g., Packed Encoding Rules) are possible.

For group III and group IV applications, specific object identifier names must be used for all the abstract and transfer syntaxes. If these names are not assigned with the specification (e.g., if the specification not in ASN.1) they can be assigned by whoever needs them - ideally the "owner" of the syntax specification.

### 3.3 Processing Presentation-context-definition-list

In Presentation context negotiation on connection establishment the initiator sends a list (the presentation context definition list) of the abstract syntaxes it intends to use, each with a list of transfer syntaxes. Each presentation context also has an integer identifier. To build the reply, a responder has to examine this list and work out which of the offered presentation contexts will be accepted and which (single) transfer syntax for each. The responder sends back the reply field, the Presentation-context-definition-result-list, in the accept message. The result list contains the same number of result items as the definition-list proposed presentation-contexts. They are matched by position, not by the identifiers (which are not present in the result-list). An acceptance also includes the transfer syntax accepted (as there can be several offered). This can be copied from the definition list.

For the group I, group II and group III cases, only the ACSE and one application-data P-context will be used and all other contexts rejected. For the group IV case, several presentation contexts will be accepted.

However, even for group I applications there may be synonyms for an application-data Presentation-context. Unknown synonyms are rejected. The reply shown in 6.2 includes a rejection (It can therefore not be the reply to the connection request shown in 6.1, since that has only two items in the definition list.)

In all cases, the connection responder must identify and keep the presentation context identifiers (called pcid's here) for all the accepted presentation contexts. These are integers (odd integers, in this case). CULR-1 limits them to no greater than 32767, but they will usually be  $\leq 255$  (so taking up one octet).

A presentation context is sometimes used (i.e., data is sent using it) before the negotiation is complete. As will be seen in section 6, in these cases, the transfer syntax name sometimes appears with the integer identifier.



### 3.4 Application context

The Association Control Service Element also exchanges the name (another Object Identifier) of the application context, which identifies what the communication is all about, again independently of the naming and addressing. As for the syntaxes, although some name must be present in the protocol, a generic name can be used for applications that do not have a specific name assigned. (This will almost certainly be a group I application - if a specific name is assigned, it must be used.) The only negotiation allowed is that the reply may be different from that sent by the initiator. CULR-3 provides a generic application context name (i.e., assigns an object identifier).

### 3.5 APtitles and AEqualifiers

In addition to the addressing constructs (transport address and possibly session and presentation selectors), the communicating application entities have names - Application-Entity titles (AETitle). These are carried by ACSE as two fields -the Application-process titles (APTtitle) and the Application-entity qualifier (AEqualifier). The AETitle is compound, and the APTtitle consists of all but the last element, which is the AEqualifier. (This explanation can be run backwards). There are two non-equivalent forms. AP-titles and AE-titles can be Directory Name or an Object Identifier. AE-qualifiers can be Relative Distinguished Name (RDN) or an integer - the forms must match, since the AE-qualifier is the last component of the AP-title. In practice, the Directory form is likely to be the only one seen for a while.

Use of these names is rather variable. This cookbook proposes that implementations should be able to handle any value for the partner's names, and set (as initiator) its own names. This is primarily to facilitate OSI:non-OSI relaying (e.g., X/osi:X/tcp), allowing the names of the end-system to be carried to the relay, where they can be converted into hostnames, and the lower-layer address determined. OSI assumes that name-to-address lookup is possible (via the Directory or other means), but does not assume address-to-name will work. Thus the calling AE-title is needed if the responder is to know who the initiator is. However, most protocols work perfectly well without these names being included.

As for their encoding, a RDN will almost always be a single attribute value assertion, with the attribute defined either by the Directory standard [ISO9594 = X.500], or in [Internet/Cosine Schema] [RFC1274]. Using the notation defined below, the encoding of an RDN using a Directory-defined standard attribute is:

```

31 80 {1          - RDN, [SET OF]
30 80 {2          - AttributeValueAssertion, [SEQUENCE]
06 03 5504yy      -- OID identifying an attribute named in
                  -- the Directory standard
                  -- which one is determined by yy
13 La xxxxxxxx    -- [Printable string]
                  -- could be T61 string, with tag 14
00 00 }2          - end of AVA
00 00 }1          - end of RDN

```

The most likely attributes for an RDN have the following hex values for yy.

|                  |    |
|------------------|----|
| CommonName       | 03 |
| Country          | 06 |
| Locality         | 07 |
| State/Province   | 08 |
| Organisation     | 0A |
| OrganisationUnit | 0B |

For non-Directory attributes, the object id name must be substituted (thus changing the immediately preceding length)

If there are multiple attribute value assertions in the RDN, the group between {2 and 2} is repeated (with different attributes). Order is not significant.

The encoding of a [Directory] Name for the AP-titles is the RDNs (high- order first) within

```

30 80 {1          - [SEQUENCE] Name
-- put the RDN encodings here
00 00 }1

```

An Object Identifier AP-title is encoded as a primitive (see below), with the "universal" tag for an object identifier, which is 6. The integer AE-qualifier uses the universal tag for an integer, which is 2.

#### 4. What has to be sent and received

##### 4.1 Sequence of OSI protocol data units used

OSI defines its facilities in terms of services but these are abstract constructs (they do not have to correspond to procedure calls) - the significant thing is the transmission of the resulting protocol data unit (PDU). The PDU at each layer carries (as user data) the PDU of the layer above. The different layers follow

different conventions for naming the pdus. This section gives an overview of the sequence of PDUs exchanged - the details of these are given in section 6.

The requirements of the application are to create a connection (strictly an association for the application-layer in OSI, but called a connection here), to send and receive data and to close the connection. The PDUs used are thus:

To create connection:

First create transport-level connection

Initiator sends the message defined in 6.1, which is Session CONNECT carrying Presentation CONNECT request [CP] carrying ACSE A-ASSOCIATE request [AARQ] optionally carrying application data.

Responder replies with the message defined in 6.2, which is Session ACCEPT carrying Presentation CONNECT response [CPA] carrying ACSE response [AARE] optionally carrying application data.

- If the responder rejects the attempt, the reply will be Session REJECT. This is defined in 6.3, where the REJECT carries no user data. A received REJECT may carry Presentation, ACSE and application data, although 6.3 shows only how to reject at Session level..

To send/receive data on an connection

send the message defined in 6.4, which is an empty Session GIVE-TOKEN followed by Session S-DATA carrying Presentation P-DATA [TD] containing the application data (The GIVE-TOKEN is just two octets required by Session for some backwards compatibility.)

To close connection gracefully

One side sends the message defined in 6.5, which is Session FINISH carrying P-RELEASE request carrying A-RELEASE request [RLRQ] optionally carrying application data (This side may now receive, but not send data.)

The other side replies with the message defined in 6.6, which is Session DISCONNECT carrying P-RELEASE response carrying A-RELEASE response [RLRE] optionally carrying application data

First side disconnects transport connection on receiving the reply

To close connection abruptly but also send application data

Send the message defined in 6.7, which is Session ABORT carrying Presentation U-ABORT [ARU] carry ACSE U-ABORT [ABRT] carrying application data (delivery not guaranteed)

On receiving Session ABORT, disconnect transport

To close connection abruptly

- Either send the message defined in 6.8, which is Session ABORT carrying nothing;

Or, just disconnect at transport level

A group I application is assumed (by definition) not to send data on the establishment and release exchanges, a group II application will.

It would be possible to use the abort-with-data facility with a group I to send a (possibly non-standardised) error message for diagnostic purposes.

A special rule is used if a release collision occurs (i.e., FINISH+P-RELEASE+RLRQ received after sending the same): the side that initiated the original upper-layer connection waits and the other side replies with the DISCONNECT etc.

#### 4.2 Which OSI fields are used

There are a number of fields (parameters) in the pdus involved. These can be categorised by what is needed to support applications (of a particular Group) in general - a field may be "useful", "send-only", "fixed", "fixed with default", "internal" or "not important". Even those that are not important may be received from another implementation, but since the application has no use for them, they can be ignored. If an implementation is intended to support only a particular application, it may be able to downgrade the "useful" to "not important".

The text below describes the processing that is required for each category and which fields are in each category.

"Useful" - when sending, an implementation of general use should be able to set any (legal) value of these fields (via the upper interface from the application or via some configuration or lookup

mechanism) and SHOULD pass received values for the Calling values to the application (for specific applications, these fields may be either required or unnecessary.)

AARQ:

Called application-process title  
Called application-entity qualifier  
Calling application-process title  
Calling application-entity qualifier

"Send-only" - to interwork, the implementation must be able to set any value of these, but can ignore any received value. Both are octet strings.

Presentation selector (up to 4 octets, limited by CULR-1)  
Session selector (up to 16 octets, limited by base standard)

"Fixed" (constant for all applications)

abstract and transfer syntax identifiers for presentation context  
for ACSE Version numbers - 2 for session, 1 for Presentation  
and ACSE

"Fixed with default" - the value is specific to the application. For non-ASN.1 abstract syntaxes (group I or group II only) applications, the anonymous values assigned by the OIW minimal OSI profile [CULR-3] can be used. The CULR-3 default application context can be used where a proper context name is neither available nor needed.

Application context

CULR-3 default is {1 0 11188 3 3}

Abstract syntax identifier for application data

CULR-3 anonymous name is {1 0 11188 3 1 1}

Transfer syntax identifier for application data

CULR-3 anonymous name is {1 0 11188 3 2 1}

"Internal" - an arbitrary value can be sent; a received value must be stored for use in sending.

Presentation context identifiers for ACSE and the application  
data (always odd integers)

"Not important" - for interworking, any legal received value for the other fields must be received (i.e., the pdu is parsed successfully), but can then be ignored. There is no requirement (in this cookbook) to check the existence, value or internal format of these fields.

All other fields (which includes a large number of session fields)

#### 4.3 Encoding methods and length fields

Both Session and ASN.1/BER [ISO8824, ISO8825] use a type-length-value structure for their encodings, but different ones. Presentation protocol and ACSE protocol use the ASN.1/BER encoding and consequently a Presentation PDU containing an ACSE PDU can be constructed or parsed as if it were a single structure.

All the protocols contain pdu fields with a compound structure. If one of these is being ignored it may be necessary (for BER, not session) to determine the lengths of its components to find the length of the ignored field.

Many of the lengths in the specification below will vary, dependent on the values of the fields.

##### 4.3.1 Session items

The type field of a session item is always a single octet.

For session items, given a particular length, there is no flexibility:

If the length is less than 255, represent as one octet

If the length is greater, represent as three octets, first is 0xFF, next two are the length, high-order octet first.

(Some "real" implementations are known to use the second encoding in all cases. This is wrong, but should only concern conformance testers.)

##### 4.3.2 ASN.1/BER items (Presentation and ACSE)

The type field for ASN.1-BER is the tag. Although it is possible for large tags (>30) to be multi-octet, there are no large tags in the protocols involved in this memo. Bit 6 (0x20) of the tag octet is 1 if the item is constructed (i.e., the value is itself one or more ASN.1 BER items) or 0 if it is primitive.

There is considerable flexibility, at senders option, in how lengths are represented in BER. There are three forms: short, long and indefinite.

Short (usable only if the length is less than 127) : one octet

Long (usable for \*any\* length) : first octet has the top bit set, the rest is a count of how many octets are holding the length value; that many subsequent octets hold the length. A long length may use more than the minimum number of octets (so 0x8400000001 is a valid representation of length 1)

Indefinite (usable only for the length of a compound field) : the single octet is 0x80, then one or more items (their tag-length-values) and finally two octets of 0x00 (equivalent to tag and length of zero).

To be able to interwork generally, an implementation must be able to handle any of these forms when receiving.

The encodings specified in the octet sequences below use indefinite length for all constructed items with a few exceptions. This slightly increases the number of octets sent, but means that the length of a varying field (e.g., user data, or a varying object identifier) affects only the length of the item itself, and not the enclosing lengths. It is thus possible to use the octet sequences as templates interspersed by the varying fields.

It is important to note that this choice of indefinite (which is equivalent to the "Canonical Encoding Rules" variant of BER) applies only to the Presentation and ACSE protocols themselves. It does not apply to ASN.1/BER encoded application data. The processing required of application data may suggest alternative "best" options.

#### 4.4 BER Encoding of values for primitive datatypes

The following ASN.1 primitive datatypes are used in the thinosi stack.

Integers are encoded in twos-complement, high-order first. Unlike lengths, they must be encoded in the minimum number of octets (no leading 00 padding).

Object Identifiers have a rather peculiar, but compressed encoding:

Combine the first two integers of the OID into one element by multiplying the first (always 0, 1 or 2) by 40, and add the second.

Each element (that one, and each subsequent integer in the OID taken on its own), is taken as a binary number and divided into 7-bit "bytes". This is apportioned into bits 1-7 of the minimum number of octets. Bit 8 is one for all octets of the sequence except the last. (This means that elements of less than 127 are

single octet integers.)

Printable Strings - as if in ISO 646 (ASCII)

OCTET STRING - just put the octets there

#### 4.5 Unnecessary constructed encodings

BER allows the sender to break some items (such as OCTET STRINGS, character strings) into several pieces (i.e., as constructed encoding) or send them as primitive. CULR-1 requires that this is only done to one level. The pieces of both OCTET STRING and character string are tagged as if they were OCTET STRING - they have the tag 04. This memo does not include any of these optional constructions, but they may be received in interworking.

#### 5. Notation

The constructs are shown in their tag - length - value form. All numbers are in hexadecimal. Comments are preceded by a '-' character. Multiple '-' mean the comment is more than just information.

The tag column contains one of:

single fixed octets.

\* in the tag field indicates one or more pdu fields (possibly constructed) that may be received but are not sent. If received they can be ignored.

! indicates the tag is defined elsewhere.

. is a place holder for the column.

? preceding the tag value indicates that the field is not always present - the comment will explain.

The length column contains one of

explicit value

Ls - a length according to session rules which depends on the total size of the value (usually constructed)

La - a length according to BER rules

. is a placeholder



yy is exactly one octet (i.e., one hex digit per y) holding part of the length

The value column contains one of

the hex value

xxxxxx - value of varying length (sometimes constructed)

{n - (n = number) the start of a constructed value

n - (n=number) the end of the constructed value with the corresponding number. (The number is sometimes omitted on the innermost nest of construction)

yy - as part of a value - a variable value, each y represents one hex digit

? a value, possibly constructed that may be received but is not sent. It may be ignored if received

Note that all presentation lengths may be received in one of the alternative forms. All constructed lengths are shown in indefinite form. If a received length is definite, the corresponding end item (which will be shown here as 00 00 }n) will become . . }n.

In the comments, the notation {n} refers to the constructed item bracketed by the {n, }n fields.

## 6. Octet sequences

### 6.1 Connection request message

```

      - CONNECT SPDU
0D  Ls  {1      - "SI" value for CONNECT = 13
*   Ls  ?       - Connection Identifier
05  06  {2      - Connect/Accept Item
13  01  00      - protocol options (probably mandatory)
*   Ls  ?

16  01  02      -- version number (bottom bit = v1, next bit =v2.
                --          may get offers of either or both
*   Ls  ?

14  02  0002    - Session User Requirements (functional units)
                - Id (20), length (always 2), duplex fu only.
                -- On receipt, other bits may be set
                -- check that the 2 bit is set
*   Ls  ?      - we do not send any Calling Session Selector

```

```

?34 Ls  xxxx      -- Called Session Selector (i.e., the other end's)
                  -- up to 16 octets - you must set what the other
                  -- side demands. - May be anything characters,
                  -- binary etc.
                  -- {3} disappeared in editing
C1  Ls  {4         -- User Data, Identifier=193. if length is > 512,
                  -- then identifier is 194 (hex C2) instead
- CP - P-CONNECT-RI PPDU. Everything below is in ASN.1 BER
31  80  {5         - [SET]
                  --- Mode-selector (the {6} group) could possibly
                  --- come after everything else {7}
                  --- This will probably only be done by
                  --- evil-minded conformance testers
A0  80  {6         - Mode-selector [0] IMPLICIT SET
80  01  01         - [0] IMPLICIT INTEGER {normalmode(1)}
00  00  }6
A2  La  {7         - [2] unnamed IMPLICIT SEQUENCE
*   La  ?
?82 La  xxxx      - [2] Called-presentation-selector
                  - CULR says maximum length is 4
                  -- must be what the other side wants
A4  80  {8         - [4] Presentation-context-definition-list
                  --- items (the outer SEQUENCES) within the {8} list may
                  --- be in any order.
30  80  {9         - [SEQUENCE]
02  01  01         -- Defines pcid for ACSE; received value will be
                  -- a one or two octet odd integer
06  04  52010001 - [OID] for ACSE abstract syntax
30  80  {          - [SEQUENCE]
06  02  5101      - [OID] Transfer syntax name is BER
00  00  }          - end t-s list
00  00  }9         - end acse pctx defn
30  80  {10        - [SEQUENCE]
02  01  03        -- [INTEGER] Defines pcid for application data;
                  -- received value will be a one or two octet odd
                  -- integer
06  La  xxxxxxxx - [OID] object identifier name of application
                  - abstract syntax (if CULR-3 default is used, this
                  - line is 06  06  28D734030101)
30  80  {11
06  La  xxxxxxxx - [OID] t-s name for application data
                  - (if CULR-3 default is used, this line is
                  - 06  06  28D734030201)
                  -- will be several of these if multiple t-s offered
                  -- (application is Group III)
                  -- all will have the same tag 06
00  00  }11        - end transfer syntax list for application p-ctx
00  00  }10        - end application pctx definition

```

```

-- if multiple presentation contexts are offered, (Group
-- IV), the {10} SEQUENCE will repeat appropriately
-- if multiple contexts are to be accepted, all the
-- pcid's must be remembered
00 00 }8      - end of p-ctx-def-list
*   La ?
61 80 {12     - [APPLICATION 1] User-data - Fully-encoded
30 80 {13     - [SEQUENCE] PDV-list
02 01 01     -- [INTEGER], value is acse pcid
A0 80 {14     - [0] Single-ASN1
- ACSE A-ASSOCIATE request APDU - AARQ
60 80 {15     - [APPLICATION 0] - AARQ
*   La ?      - protocol version defaults to 1 (only one defined)
A1 80 {       - [1] Application-context
06 La xxxxxxx -- object identifier name of application context
                - (if CULR-3 default is used, this line is
                - 06 05 28D7340303)

00 00 }
-- Called application process title {16} and application
-- entity qualifier may or may not be needed (see 3.4)
?A2 80 {16    - [2] Called Application-Process title
?! La xxxxxxx -- see 3.5 - either a Directory Name or an oid
?00 00 }16    - end Called APTitle
?A3 80 {17    - [3] Called Application-Entity Qualifier
?! La xxxxxxx -- see 3.5
?00 00 }17
*   La ?
        Calling AP-title and AE-qualifier may or may not be needed.
?A6 80 {18    - [6] Calling Application-Process title
?! La xxxxxxx -- see 3.5
?00 00 }18
?A7 80 {19    - [7] Calling Application-Entity Qualifier
?! La xxxxxxx -- see 3.5
?00 00 }19
*   La ?
        -- the user information field may or may not be required
        -- (not required for Group I)
?BE 80 {20    - [30] IMPLICIT SEQUENCE
?28 80 {21    - [EXTERNAL]
?06 La xxxxxxx -- [OID] This is the oid identifying the transfer
                -- syntax used for the user data.
                -- It is (almost certainly) required even if only
                -- one transfer syntax was proposed.
?02 01 03     - [INTEGER] this is the pcid for the application
                - data
?A0 La xxxxxxx -- [0] single-ASN.1-type - the application data
                -- (see paragraph at end of this section below)
?00 00 }21    - end of EXTERNAL

```

```

        -- conceivably there may be several EXTERNALS, probably in
        -- different presentation contexts (different pcids)
?00 00 }20      - end of user information field
00 00 }15      - end of AARQ
00 00 }14      - end of single-ASN-type
00 00 }13      - end of PDV-list
00 00 }12      - end of Presentation User-data
00 00 }7       - end of third element of CP-type SET
00 00 }5       - end of CP-type

```

The application data carried in the EXTERNAL is shown (as A0 La xxxx) assuming it is a single-ASN.1 type, which it often will be for group II (since these tend to be OSI applications). The xxxx will be the BER encoding of the application pdu (probably something like Z-BIND or Y- INITIALIZE). The length may be indefinite.

If the application data is not a single ASN.1 type, but is an octet-aligned value, the A0 La xxxx is replaced by 81 La xxxx, where xxxx is the value. In this case the length must be definite (unless an "unnecessary" constructed encoding is used.)

Identical considerations apply to the other EXTERNALs carried in the ACSE pdus.

## 6.2 Successful reply to connection setup

If the connection attempt is successful, the following is sent to the initiator on a T-DATA.

```

0E  Ls  {1      - ACCEPT SPDU
*   Ls  ?
05  06  {2      - Connect/Accept Item
13  01  00      - Protocol Options
*   Ls  ?
16  01  02      - version number (this shows version 2 only)
        -- if version 2 was not offered, omit all of {2}
*   Ls  ?
14  02  0002    - Session User Requirements (functional units)
        - duplex fu only (kernel is automatic)
*   Ls  ?
C1  Ls  {3      -- User Data.
- CPA - P-CONNECT response
31  80  {4      - [SET]
        -- again, Mode-selector could come at the end
A0  80  {      - Mode-selector [0]
80  01  01      - normal mode - [0], length=1, value=1
00  00  }
A2  80  {5      - [2] SEQUENCE (unnamed)

```

```

*   La   ?
A5  80   {6      - [5] P-context-definition-result-list
                  -- following result items are in the order
                  -- corresponding to the pctx-definition-list in
                  -- the connect
                  -- this example assumes that was ACSE, user, rubbish
                  -- with rubbish rejected
30  80   {7      - [SEQUENCE] result item for acse
80  01   00      -- [0] result, value 0 is acceptance
81  02   5101    - [1] accepted transfer syntax name = BER
                  - note that this has an implicit tag, not 06
00  00   }7      - end result item for acse p-ctx

30  80   {8      - [SEQUENCE] result item for application-data pctx
80  01   00      - [0] value 0 is acceptance
81  La   xxxxxxx - [1] oid for transfer syntax, as on definition list
                  -- if there were several (groupIII) , the one you
                  -- liked most
00  00   }8      - end result item for app-data p-ctx
00  00   }6      - end p-ctx-def-result-list
*   La   ?
61  80   {10     - [APPLICATION 1] User-data, Fully-encoded

30  80   {11     - [SEQUENCE] PDV-list
02  01   01      -- [INTEGER] value is pcid for ACSE, as stored from
                  -- the pctx-definition-list on the P-CONNECT
                  -- request
A0  80   {12     - [0] single-ASN1-type
                  - A-ASSOCIATE response APDU - AARE
61  80   {13     - [APPLICATION 1] identifies AARE
*   La   ?
A1  80   {14     - [1] Application-context
06  La   xxxxxxx - [OID] name of application context
                  - usually the same as on AARQ, can differ

00  00   }14
A2  03   {15     - [2] result
02  01   00      - [INTEGER] value 0 means accepted
00  00   }15
A3  80   {16     - [3] result-source-diagnostic
                  - (curiously, a non-optional field)
A1  80   {17     - [1] acse-service-user
02  01   00      - [INTEGER] value 0 = null ! (why no implicit tag)
00  00   }17     - end acse-service-user
00  00   }16     - end result source diagnostic
*   La   ?
                  -- the user information field may or may not be required
                  - (not used for Group I)
?BE 80   {20     - [30] IMPLICIT SEQUENCE

```

```

?28 80 {21      - [EXTERNAL]
              -- the transfer-syntax oid is not present this time
?02 01 03      - [INTEGER] this is the pcid for the application
              - data
?A0 La xxxxx   -- [0] single-ASN1-type (see note at end of 6.1)
?00 00 }21      - end of EXTERNAL
              -- conceivably there may be several EXTERNALS, probably in
              -- different presentation contexts (different pcids)
?00 00 }20      - end of user information field
00 00 }13      - end AARE
00 00 }12      - end single-asn1-type
00 00 }11      - end PDV-list
00 00 }10      - end Presn user-data
00 00 }5       - end [2] implicit sequence in cpa
00 00 }4       - end CPA-type set

```

The following sequence are the octets need to reject a presentation context that was offered in the presentation-context-definition-list. Since the result-list matches the definition list by position, it is placed at the corresponding point within {6} (e.g., it would come immediately after {8}, if the rejected context was the third one.

```

              -- next SEQUENCE is a rejection of a pctx
30 80 {9       - [SEQUENCE] result item for a rejected pctx
80 01 02      -- [0] result, value 2 is provider rejection
82 01 00      - [2] reason, value 0 is reason-not-specified
              -- there are other reasons, but let's keep it
              -- simple
00 00 }9       - end result item for rejected pctx

```

### 6.3 Connection rejection

Refusal is at session-level, but by session user, with no reason given. This is a compromise avoiding making unfounded accusations of (session) protocol misbehaviour. If the implementation finds it does not like the received message, it is not essential to attempt to communicate with the partner why, though this may be helpful if the reason is correctly identified. (In most cases, a wise implementor will make sure an error message goes somewhere or other).

```

0C 03 {1       - REFUSE SPDU
*   Ls ?
32 01 00      - rejected by SS-user, no reason

```

The far-end may send interesting things explaining why you are not getting interworking. If this is a session reason, the reason code will one octet between 81 and 86. If the rejection is higher than session, this will be carried on S-REFUSE (so first octet is still

0C) and the higher pdu will appear as part of the reason code, which will start with 02. (The only remaining code is 01 = user congestion.)

#### 6.4 Data-phase TSDU

This is the core of the skinny stack. The lengths shown use a particular set of choices for indefinite and definite lengths that means that the application data length only affects one field. Making the two earlier indefinite lengths definite would require more calculation - adding 4 octets after the application data is assumed to be quicker. This header is also designed to be 20 octets long, thus maintaining 4-byte alignment between transport and application buffers. Implementations are recommended to use this encoding. It is possible to rapidly match incoming data against it - if there is no mismatch until the length field, the location of the beginning of the data can be determined without further analysis.

```

          SPDUs
01  00  .      - S-GIVE-TOKEN - required by basic concatenation
                  - but no parameters
01  00  .      - S-DATA - no parameters - what follows is User
                  - Information, not User Data, so is not included in
                  - the SPDU length fields
- P-DATA PPDU - TD (why TD ? Typed-data id TTD !)
61  80  {1      - User-data [APPLICATION 1]
30  80  {2      - [SEQUENCE] PDV-list
02  01  03      - [INTEGER] pcid for application data, P-CONNECT PPDU
                  - remembered by both sides
81  83yyyyyy    xxxxxx -- [1] octet-aligned presentation data value(s)
                  -- length of length (3 octets) then three octets yyyyyy
                  -- for the length of the user data xxxxxx
00  00  }2      - End-of-contents for end of PDV-list
00  00  }1      - End-of-contents for end of Presentation User-data

```

If the application data is in ASN.1, and a single ASN.1 value is being sent on the TSDU, the same header can be used except for the tag on the presentation data values, which becomes A0 (= [0], constructed).

If there are multiple data values to be sent, this header can be expanded in several ways:

- a) if there are several ASN.1 values from the same presentation context, they can be concatenated and treated as an octet-aligned value (using the header as shown above, with tag 81 (or A1 - I think its primitive) or each ASN.1 value can be an item

(tagged A0), one after the other

- b) if the data values are from different presentation contexts (group IV), each is in its own {2} group within the {1}.

On receipt, for the simple octet-aligned case, the data value may itself have a constructed encoding - this will make the tag A1, and it will contain elements each tagged 04 (OCTET STRING). According to CULR- 1, these elements are primitive (otherwise they would be 24 of course).

## 6.5 Closedown - release request

When all is done, and you want to close down gracefully, send this on T-DATA.

```

- FINISH SPDU
09 10 {1      - 9 identifies FINISH
*  Ls  ?      - No Transport Disconnect item
              - default is release Transport-connection
C1 0E {2      - User data (code 193)
- P-RELEASE req/ind PPDU (has no name)
61 80 {3      - [APPLICATION 1], user data, fully-encoded
30 80 {4      - [SEQUENCE] PDV-list
02 01 01      -- pcid for ACSE, remembered from setup
A0 80 {5      - [0] single asn.1-type
- A-RELEASE request APDU - RLRQ
62 80 {6      - [APPLICATION 2] identifies RLRQ
80 01 00      - [0] reason, value 0 means normal
*  La  ?
      -- the user information field may or may not be required
      - ( not required for Group I)
?BE 80 {7      - [30] IMPLICIT SEQUENCE
?28 80 {8      - [EXTERNAL]
      -- the transfer-syntax oid is not present this time
?02 01 03      - [INTEGER] this is the pcid for the application
              - data
?A0 La xxxxx   -- [0] single-ASN.1-type application data
              -- (see note at end of 6.1)
?00 00 }8      - end of EXTERNAL
      -- conceivably there may be several EXTERNALS, probably in
      -- different presentation contexts (different pcids)
?00 00 }7      - end of user information field
00 00 }6      - end of RLRQ
00 00 }5      - end of single asn.1-type
00 00 }4      - end of PDV-list
00 00 }3      - end of Presentation User-data

```



## 6.6 Closedown - release response

On receiving a FINISH, you send this to tell the other end it is all over

```

    - Session DISCONNECT SPDU
0A  Ls  {1          - SI=10, DISCONNECT
C1  Ls  {2          - User data
    - P-RELEASE rsp PPDU
61  80  {3          - [APPLICATION 1], user data, fully-encoded
30  80  {4          - [SEQUENCE] PDV-list
02  01  01          -- [INTEGER] pcid for ACSE, remembered from setup
A0  80  {5          - [0] single asn.1-type
    - A-RELEASE response APDU - RLRE
63  80  {6          - [APPLICATION 3] identifies RLRE
80  01  00          - [0] reason, value 0 means normal
*   La  ?
        -- the user information field may or may not be required
        - (not required for Group I)
?BE 80  {7          - [30] IMPLICIT SEQUENCE
?28 80  {8          - [EXTERNAL]
        -- the transfer-syntax oid is not present this time
?02 01  03          - [INTEGER] this is the pcid for the application
        - data
?A0 La  xxxxx       -- [0] single-ASN.1-type application data
        -- (see note at end of 6.1)
?00 00  }8          - end of EXTERNAL
        -- conceivably there may be several EXTERNALS, probably in
        -- different presentation contexts (different pcids)
?00 00  }7          - end of user information field
00  00  }6          - end of RLRE
00  00  }5          - end of single asn.1-type
00  00  }4          - end of PDV-list
00  00  }3          - end of Presentation userdata

```

## 6.7 Deliberate abort

It is not clear whether this is any use - just clearing the Transport connection will be more effective. It goes on T-DATA, but asks for the far-side to close the T-connection.

```

    - Session ABORT SPDU
19  Ls  {1          - SI of 25 is ABORT
11  01  03          - Transport Disconnect PV, code 17
        -- value = '...00011'b means please
        -- release T-conn, user abort
*   Ls  ?
C1  11  {2          - Session User Data

```

```

- P-U-ABORT PPDU - ARU
A0 80 {3 - [0] implicit sequence for normal mode
A0 80 {4 - [0] presentation-context-identifier-list
30 80 {5 - [SEQUENCE]
02 01 01 - [INTEGER]pcid for ACSE
06 02 5101 - [OID] for acse transfer syntax = BER
00 00 }5
-- there will be one {6} group for each application
-- presentation context that is used in this message
-- if there is no user data, the {6} group can be
-- omitted
30 80 {6
02 01 03 - [INTEGER] pcid for application data
06 La xxxxxxx - [OID] transfer syntax for application data
00 00 }6
00 00 }4 - end of presentation-context-identifier-list
61 80 {7 - [APPLICATION 1], user data, fully-encoded
30 80 {8 - [SEQUENCE] PDV-list
02 01 01 - [INTEGER] pcid for ACSE as on CP PPDU
A0 05 {9 - [0] single asn.1-type
- A-ABORT APDU - ABRT
64 80 {10 - [APPLICATION 4] identifies ABRT
80 01 01 - [0] value 1 is acse-service-provider
-- the user information field may or may not be required
?BE 80 {11 - [30] IMPLICIT SEQUENCE
?28 80 {12 - [EXTERNAL]
-- the transfer-syntax oid is not present this time
-- (according to CULR-1)
?02 01 03 - [INTEGER] this is the pcid for the application
- data
?A0 La xxxxxx -- [0] single-ASN.1-type application data
-- (see note at end of 6.1)
?00 00 }12 - end of EXTERNAL
-- conceivably there may be several EXTERNALS, probably in
-- different presentation contexts (different pcids)
?00 00 }11 - end of user information field
00 00 }10 - end of ABRT
00 00 }9 - end of single asn.1-type
00 00 }8 - end of PDV-list
00 00 }7 - end of Presentation user-data
00 00 }3 - end of ARU-PPDU

```

## 6.8 Provider abort

Generated when an internal error occurs (i.e., something has gone mildly (?) wrong in the cookbook implementation). Rather than accuse anyone of protocol errors, we just abort at session.

```

      ABORT SPDU
19  03  {1      - SI=25 = ABORT SPDU
11  01  09      - Transport Disconnect PV, code 17
              -- value = '...01001'b  release T-conn
              -- no reason
*   Ls   ?

```

## 6.9 Abort accept

If a Session abort (of any kind) is sent, it is possible that the far-end will send back an abort accept. If this happens, disconnect the transport. (The abort messages above do not propose that the transport connection be reused, and in this case, an abort accept is just the far-end passing the transport-disconnect initiative back.) This session message need never be sent - just disconnect transport on receiving an abort.

```

      ABORT ACCEPT SPDU
1A  00  .      - SI=26 = ABORT ACCEPT SPDU, no parameters

```

## 7. References

[CULR-1] ISO/IEC DISP 11188-1 - Information Technology - International Standardised Profile - Common Upper Layer Requirements - Part 1: Basic Connection oriented requirements (DISP ballot ends June 1994).

[CULR-3] Draft of Common Upper-layer requirements - Part 3: Minimal OSI upper layers facilities (A later draft will be proposed as ISP 11188/3).

[ISO8072] Information processing systems - Open Systems Interconnection - Transport service definition; ISO, 1986.

[ISO8073] Information processing systems - Open Systems Interconnection - Transport protocol specification; ISO, 1986.

[ISO8326] Information processing systems - Open Systems Interconnection - Basic connection oriented session service definition; ISO, 1987 (or review copy of revised text = ISO/IEC JTC1/SC21 N4657, April 1990).

[ISO8327] Information processing systems - Open Systems Interconnection - Basic connection oriented session protocol specification; ISO, 1987 (or review copy of revised text = ISO/IEC JTC1/SC21 N4656, April 1990).

[ISO8649] Information processing systems - Open Systems Interconnection - Service definition for the Association Control Service Element; ISO, 1989.

[ISO8650] Information processing systems - Open Systems Interconnection - Protocol specification for the Association Control Service Element; ISO, 1989.

[ISO8822] Information processing systems - Open Systems Interconnection - Connection-oriented presentation service definition; ISO, 1989.

[ISO8823] Information processing systems - Open Systems Interconnection - Connection-oriented presentation protocol specification; ISO, 1989.

[ISO8824] Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), ISO/IEC 1990.

[ISO8825] Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One, ISO/IEC 1990.

[RFC1006] Rose, M., and D. Cass, "ISO Transport Services on Top of the TCP", STD 35, RFC 1006, Northrop Research and Technology Center, May 1987.

[ISO9594] Information technology - Open Systems Interconnection - The Directory; ISO/IEC, 1990.

[RFC 1274] Barker, P., and S. Kille, "The COSINE and Internet X.500 Schema", RFC 1274, University College London, November 1991.

## 8. Other Notes

The Session, Presentation and ACSE standards have been the subject of considerable amendment since their first publication. The only one that is significant to this cookbook is Session addendum 2, which specifies session version 2 and unlimited user data. New editions of these standards, incorporating all the amendments, will be published during 1994.

The coding choices made in the cookbook are (nearly) those made by the "Canonical Encoding Rules", which are a form of Basic Encoding Rules with no optionality, specified in the new edition of ISO/IEC 8825. A defect report has been proposed against Presentation and ACSE, suggesting that a note to the protocol specifications recommend use of the canonical encoding options when sending, and then optimising for this on receipt.

## 9. Security Considerations

Security issues are not discussed in this memo.

## 10. Author's Address

Peter Furniss  
Peter Furniss Consultants  
58 Alexandra Crescent  
Bromley, Kent BR1 4EX  
UK

Phone & Fax +44 81 313 1833  
EMail: P.Furniss@ulcc.ac.uk

