Systems Integration Specialists Company, Inc.
6605 19½ Mile Road
Sterling Heights, MI 48314-1408 USA
Phone: +810-254-0020 Fax: +810-254-0053
E-Mail: info@sisconet.com, URL: http://www.sisconet.com

MMS and ASN.1 Encodings

Simple Examples and Explanations on How to Crack an MMS PDU

Created by: Herbert Falk (SISCO) and Dr. Martin Burns (Hypertek)

Date: 06/07/96
Revised: 05/30/97
08/29/01

Table of Contents

Introduction

During my work with the electrical utility industry, several educational issues in regards to MMS have often been discussed. The issues of understanding the ISO/IEC-9506 standard have always been a core problem for the non-initiated.

For the MMS non-initiated, the MMS standard is a problem since it infers the use of other standards (e.g., Presentation, ASN.1, and ACSE). Besides not being a self-contained standard, the division of protocol specification and encoding represents a different philosophy from many of the previous SCADA protocols. However, given all the required documentation, it still takes considerable time to understand ASN.1 and MMS.

This document is intended to help boot-strap individuals in the educational process. In order to achieve an initial understanding of the encoding of MMS, I have intentionally restricted the included examples of commonly used semantics and protocols.

Abstract Syntax Notation 1 (ASN.1)

The purpose of ASN.1 is to provide encoding and decoding specifications for protocol syntax that is to be sent over a network. The intent of this standard (ISO/IEC 8824 and 8825) is to have a neutral representation of fields as they are exchanged over a communication media.

Therefore, ASN.1 accounts for the problems typically associated in exchanging data between Intel, Motorola, VAX, and RISC platforms. This includes accounting for the Big/Little Endian problems and byte representation issues.

In order to accomplish this, ASN.1 concretely specifies the sequence and order of bits as they are to be transmitted on the wire. Additionally, the standard defines key words that are to be used within MMS, and other standards, to aid in the specification of the semantics and encodings. For example, ASN.1 always typically specifies that the most significant bit of the most significant byte is encoded to be transferred first.
The key words, found in MMS, are all capitalized. For example:

SEQUENCE	SEQUENCE OF
IMPLICIT	INTEGER
BOOLEAN	NULL

Also, ASN.1 encoded values always have the format of TAG, LENGTH, followed by VALUE. As with any rule, there is typically an exception. In the case of ASN.1, it is NULL, which only has a TAG and always has a length of 0 and no data (what better way to represent a NULL value). The combination of a TAG, LENGTH, and VALUE (TLV) is termed a “production”.

A **TAG** describes the ASN.1 kind of information represented by the production.

A **LENGTH** describes how many bytes (OCTETS in ASN.1 speak) that follow in the **DATA** part of the production.

A **VALUE** is the actual content being carried by the production.

TAGS

The TAG byte has several sub-fields designated. These are:

Bits 7,6	Type of tag
Bit 5	Primitive or Constructed Flag
Bit 4-0	Tag value

For MMS, the values of bits 7-5 that are typically used are:

Bits 7,6	Bit 5	Description, key words
0 0	0	Description: Universal Tag, Primitive Example keywords: INTEGER, BITSTRING, BOOLEAN
0 0	1	Description: Universal Tag, Constructed Example keyword: SEQUENCE, SEQUENCE OF
1 0	0	Description: Context Specific, Primitive Example Keyword: IMPLICIT
1 0	1	Description: Context Specific, Constructed Example Keywords: IMPLICIT SEQUENCE IMPLICIT SEQUENCE OF

The other commonly encountered keyword is CHOICE, which has no encoding.

The actual tag values are assigned in ASN.1 (for primitive Universal tags) or via the MMS standard through the notation using '[]'. The integer value within the '[]' denotes the actual value of the tag to be used.

The Universal tag values, defined within the ASN.1 standard are:

Keyword	Tag Value (hex)
BOOLEAN	01
INTEGER	02
BITSTRING	03
OCTETSTRING	04
NULL	05
OBJECT IDENTIFIER	06

Keyword	Tag Value (hex)
SEQUENCE	10
IA5STRING	16
UTCTIME	17
GENERALIZETIME	18
VISIBLESTRING	1A

The use of IMPLICIT is often misunderstood, but it is used to save bytes that are encoded and transmitted over the wire. The following example illustrates this fact:

Example: [1] INTEGER

This would encode as A1 xx 02 yy where xx and yy are lengths

In this case the [1] represents an EXPLICIT TAG (IMPLICIT SEQUENCE) and therefore is encoded as A1.

vs.

[1] IMPLICIT INTEGER ::= 81 xx

From a decoding perspective, the two methods convey the same information but IMPLICIT requires two(2) less bytes.

The Tag value (bits 4-0) is extensible so that tag values greater than 31 can be encoded. This is accomplished by reserving 0x1f (bits 4-0), in the first tag byte, to indicate that the tag field is being extended. In general, if 0x1f is encountered in the first tag byte, the next byte is the actual tag value.

LENGTH

The LENGTH has a simple and extended form. If the length of the DATA is less than 128 bytes, the LENGTH is that number of bytes. If the length of the DATA is greater than or equal to 128 bytes, the LENGTH is encoded as several bytes. The first indicating how many bytes encode the actual length, and with bit 7 set. The subsequent length bytes contain the actual length.

Example: OCTETSTRING which is 127 bytes in length would have a TLV of:

04 7f <127 bytes of value>

Example: OCTETSTRING which is 256 bytes in length would have a TLV of:

04 82 01 00 <256 bytes of value>

DATA

The DATA portion of the production contains the actual information to be exchanged. The content is described by the TAG, discussed above, and can consist of simple types and constructed types. The constructed types -- sets and sequences-- will be comprised of separate productions representing the individual elements contained (nested) in the DATA part of the constructed production.

ASN.1 specifies the encoded form of the value for a given ASN.1 TAG.

ASN.1 example encodings

The following are ASN.1 encoding examples:

Example: INTEGER

> This would be encoded as a primitive Universal tag. Therefore, the tag value is found in the ASN.1 specification and is 02 (hex).

Example: [5] IMPLICIT INTEGER.

> Since the keyword IMPLICIT is displayed, the actual tag value for the INTEGER will be encoded as a primitive context-specific tag. The value of the tag to be used will be 5. Therefore, the encoded tag value would be 85 (hex).

Example: BIT STRING

> The TAG/Length/Value (TLV) format of ASN.1 is the rule. However, the VALUE portion is constrained to values as defined within the ASN.1 standard. Thus, the value of a BIT STRING is not only the value of the bits to be conveyed, but also the number of unused bits. Therefore, a BIT STRING that is 11 bits long would have the following encoding:
>
> 03 (tag) 03 (length of value in bytes) 05 (unused bits) ff 00 (bits).

MMS

Understanding the Protocol Notation

The MMS protocol specification makes use of several other conventions. These conventions are:

1. Keywords which are entirely capitalized are defined in the ASN.1 specification.

2. Keywords which begin with a lower case letter are defined on the same line as the keyword.

3. Keywords which begin with a capitalized letter and are not entirely capitalized are defined elsewhere within the MMS specification.

Example: Identify-Request (Confirmed MMS Request)

In order to understand how to encode (ASN.1) an entire Identify Request, several protocol productions are required. This example shows all of the productions required in order to construct the request and the sections in the MMS standard (Part 2) where the entire production can be found.

```
MMSpdu ::= CHOICE {                                                    --  ASN.1
confirmed-RequestPDU  [0] IMPLICIT Confirmed-RequestPDU,               -- section 7.1
                      ...... }

Confirmed-RequestPDU ::= SEQUENCE {                                    -- ASN.1
        invokeID            Unsigned32,                                        --
section 7.6.2
        ConfirmedServiceRequest                                        -- section 7.5.2
        }

Unsigned32 ::= INTEGER with range restrictions                         -- ASN.1

ConfirmedServiceRequest ::=  CHOICE {                                  -- ASN.1
                                    .....
        [2] IMPLICIT Identify-Request,                                 --  section 9.5
                                    ......}

Identify-Request ::= NULL
```


The various scattered productions result in the following concrete ASN.1:

```
                                                        Tag value (hex)
Identify-RequestPDU ::= [0] IMPLICIT SEQUENCE {              A0
            INTEGER,                                    02
            [2] IMPLICIT NULL                           82
                  }
```

The fact that the protocol productions are not in one section can make MMS difficult to understand, but the entire protocol always decomposes into ASN.1 defined tags.

OPTIONAL and DEFAULT Elements

When reviewing the MMS specification, the use of the OPTIONAL and DEFAULT keywords have significance.

The OPTIONAL indicates that the specified field may or may not be encoded. Additionally, it indicates that no value can be associated with the fact that the optional field is absent.

Example:

```
DefineNamedVariable-Request ::= SEQUENCE {
      variableName          [0] ObjectName,
      address               [1] Address,
      typeSpecification     [2] TypeSpecification OPTIONAL
      }
```

In this example, the request may include a typeSpecification field. However, the MMS protocol does not require this field in order to define a NamedVariable. Therefore, the field is OPTIONAL.

The use of DEFAULT indicates that the specified field may or may not be encoded. However, unlike OPTIONAL, the absence of the encoded field has significance and therefore the protocol specifies the associated value that is inferred when the field is not present.

Example:

```
GetNameList-Response ::= SEQUENCE {
listOfIdentifier[0] IMPLICIT SEQUENCE OF Identifier,
moreFollows            [1] IMPLICIT BOOLEAN DEFAULT TRUE
          }
```

In this example, the request may include a moreFollows field. However, the MMS protocol does not require this field in order to return a GetNameList-Response. If the field is not present, the receiving client can assume that all of the names have not been transferred (moreFollows=TRUE).

MMS Variable Data Values and ASN.1

MMS identified additional requirements (e.g., data precision or range of value restrictions) for Variable Data Values. This arose from the acknowledgment that ASN.1 does not explicitly differentiate between signed INTEGER values and unsigned INTEGER values.

Example: Encoding of the INTEGER value of 1 with ASN.1

INTEGER (1) ::= 02 01 01

MMS Integer(1) ::= 85 01 01

MMS Unsigned Integer(1) ::= 86 01 01

The rationale as to encoding signed/unsigned is to allow applications to convey acceptable ranges of values.

Therefore, MMS has defined its own set (Context Specific) tags to be used to convey its Variable Data values. The following table shows examples of the MMS data production and their encodings.

Common MMS Data Value Encoding Examples			
Data Type	MMS Tag (hex)	Example Value	Encoding
array	81	int [2] = {0,1]	81 06 85 01 00 85 01 01 ^ ^ ^ | | +- 1 | +----------- 0 +------------------- array

structure	82	struct {
		int 0,
		bool TRUE
		}
			82 06 85 01 00 83 01 01
			^ ^ ^
			| | + TRUE
			| +------------ 0
			+------------------- struct

boolean	83	TRUE	83 01 01
bit-string	84	1010 (bin)	84 02 04 A0
integer	85	255 -255	85 02 00 ff 85 02 80 ff
unsigned	86	255	86 01 ff
floating-point	87	1.0	87 05 08 3f 80 00 00 -- IEEE Format
octet-string	89	01 02 (hex)	89 02 01 02
visible-string	8a	"ab"	8a 02 61 62
timeofday	8c	1:00:05 1:00:05, 12/31/96	8c 04 .. .. 8c 06 .. ..
bcd	8d	09 09 09 (hex)	8d 02 03 e7 -- Integer value 999
booleanArray	8e	1010	8e 03 04 a0

MMS PDU's

The following table shows the different types of MMS PDU's. The first TAG of an MMS message is one of these values:
ASN.1:

```
MMSpdu ::= CHOICE
        {
        confirmed-RequestPDU        [0]     IMPLICIT Confirmed-RequestPDU,
        confirmed-ResponsePDU       [1]     IMPLICIT Confirmed-ResponsePDU,
        confirmed-ErrorPDU          [2]     IMPLICIT Confirmed-ErrorPDU,
        unconfirmed-PDU             [3]     IMPLICIT Unconfirmed-PDU,
        rejectPDU                   [4]     IMPLICIT RejectPDU,
        cancel-RequestPDU           [5]     IMPLICIT Cancel-RequestPDU,
        cancel-ResponsePDU          [6]     IMPLICIT Cancel-ResponsePDU,
        cancel-ErrorPDU             [7]     IMPLICIT Cancel-ErrorPDU,
        initiate-RequestPDU         [8]     IMPLICIT Initiate-RequestPDU,
        initiate-ResponsePDU        [9]     IMPLICIT Initiate-ResponsePDU,
        initiate-ErrorPDU           [10]    IMPLICIT Initiate-ErrorPDU,
        conclude-RequestPDU         [11]    IMPLICIT Conclude-RequestPDU,
        conclude-ResponsePDU        [12]    IMPLICIT Conclude-ResponsePDU,
        conclude-ErrorPDU           [13]    IMPLICIT Conclude-ErrorPDU
        }
```

MMS PDU	TAG
confirmed-RequestPDU	a0
confirmed-ResponsePDU	a1
confirmed-ErrorPDU	a2
unconfirmed-PDU	a3
rejectPDU	a4
cancel-RequestPDU	a5
cancel-ResponsePDU	a6
cancel-ErrorPDU	a7
initiate-RequestPDU	a8
initiate-ResponsePDU	a9
initiate-ErrorPDU	aa
conclude-RequestPDU	ab
conclude-ResponsePDU	ac
conclude-ErrorPDU	ad

ConfirmedService PDU's

Confirmed Service MMS PDUs all have the same general form:

Request/Response TAG	A0/A1	followed by length field
InvokeID	02 len ID	
ConfirmedService Tag	See table below followed by length	
Service Specific	See MMS standard.	

The following represents the TAGs and their encodings for MMS confirmed services.

MMS Confirmed Services TAG Table			
	TAG	Encoded TAG	
MMS Service	(dec)	Request	Response
status	0	80	a0
getNameList	1	a1	a1
identify	2	82	a2
rename	3	a3	83
read	4	a4	a4
write	5	a5	a5
getVariableAccessAttributes	6	a6	a6
defineNamedVariable	7	a7	87
defineScatteredAccess	8	a8	88
getScatteredAccessAttributes	9	a9	a9
deleteVariableAccess	10	aa	aa
defineNamedVariableList	11	ab	8b
getNamedVariableListAttributes	12	ac	ac
deleteNamedVariableList	13	ad	ad
defineNamedType	14	ae	8e
getNamedTypeAttributes	15	af	af
deleteNamedType	16	b0	b0
input	17	b1	91
output	18	b2	92
takeControl	19	b3	b3
relinquishControl	20	b4	94
defineSemaphore	21	b5	95
deleteSemaphore	22	b6	96
reportSemaphoreStatus	23	b7	b7
reportPoolSemaphoreStatus	24	b8	b8
reportSemaphoreEntryStatus	25	b9	b9
initiateDownloadSequence	26	ba	9a
downloadSegment	27	9b	bb
terminateDownloadSequence	28	bc	9c
initiateUploadSequence	29	9d	bd

MMS Confirmed Services TAG Table			
	TAG	Encoded TAG	
MMS Service	(dec)	Request	Response
uploadSegment	30	9e	be
terminateUploadSequence	31	9f 1f	9f 1f
DomainDownload	32	bf 20	9f 20
DomainUpload	33	bf 21	9f 21
loadDomainContent	34	bf 22	9f 22
storeDomainContent	35	bf 23	9f 23
deleteDomain	36	9f 24	9f 24
getDomainAttributes	37	9f 25	bf 25
createProgramInvocation	38	bf 26	9f 26
deleteProgramInvocation	39	9f 27	9f 27
start	40	bf 28	9f 28
stop	41	bf 29	9f 29
resume	42	bf 2a	9f 2a
reset	43	bf 2b	9f 2b
kill	44	bf 2c	9f 2c
getProgramInvocationAttributes	45	9f 2d	bf 2d
obtainFile	46	bf 2e	9f 2e
defineEventCondition	47	bf 2f	9f 2f
deleteEventCondition	48	bf 30	9f 30
getEventConditionAttributes	49	bf 31	bf 31
reportEventConditionStatus	50	bf 32	bf 32
alterEventConditionMonitoring	51	bf 33	9f 33
triggerEvent	52	bf 34	9f 34
defineEventAction	53	bf 35	9f 35
deleteEventAction	54	bf 36	9f 36
getEventActionAttributes	55	bf 37	bf 37
reportEventActionStatus	56	bf 38	9f 38
defineEventEnrollment	57	bf 39	9f 39
deleteEventEnrollment	58	bf 3a	9f 3a
alterEventEnrollment	59	bf 3b	bf 3b
reportEventEnrollmentStatus	60	bf 3c	bf 3c
getEventEnrollmentAttributes	61	bf 3d	bf 3d
acknowledgeEventNotification	62	bf 3e	9f 3e
getAlarmSummary	63	bf 3f	bf 3f
getAlarmEnrollmentSummary	64	bf 40	bf 40
readJournal	65	bf 41	bf 41
writeJournal	66	bf 42	9f 42
initializeJournal	67	bf 43	9f 43
reportJournalStatus	68	bf 44	bf 44
createJournal	69	bf 45	9f 45
deleteJournal	70	bf 46	9f 46
getCapabilityList	71	bf 47	bf 47
fileOpen	72	bf 48	bf 48
fileRead	73	9f 49	bf 49
fileClose	74	9f 4a	9f 4a

MMS Confirmed Services TAG Table			
	TAG	Encoded TAG	
MMS Service	(dec)	Request	Response
fileRename	75	bf 4b	9f 4b
fileDelete	76	bf 4c	9f 4c
fileDirectory	77	bf 4d	bf 4d

Data Access Errors

When a Read or Write of an MMS message fails, the response contains a Data Access Error Code from the following table:

Data Access Errors	Error
object-invalidated	0
hardware-fault	1
temporarily-unavailable	2
object-access-denied	3
object-undefined	4
invalid-address	5
type-unsupported	6
type-inconsistent	7
object-attribute-inconsistent	8
object-access-unsupported	9
object-non-existent	a

MMS Examples

The following examples are shown in a format that is neither entirely ASN.1 or MMS. In general, the examples show a typical MMSPdu. The MMS protocol specification (showing field names and IMPLICITS) along with the actual bytes used to encode the protocol syntax follows. The extracted field values are then shown.

Context Management

Initiate -

MMSPdu Received ::=

```
A8 25 80 02 08 00 81 01  05 82 01 05 83 01 05 A4
16 80 01 01 81 03 05 F8  00 82 0C 03 EE 19 00 18
00 02 00 00 00 FD 18
```

```
initiate-PDU ::=                                              TAG/Length     VALUE
 {
  [8] IMPLICIT SEQUENCE                                       A8 25
    {
     [0] IMPLICIT localDetailCalling,                         80 02          8 00
     [1] IMPLICIT proposedMaxServOutstandingCalling,          81 01          05
     [2] IMPLICIT proposedMaxServOutstandingCalled,           82 01          05
     [3] IMPLICIT proposedDataStructureNestingLevel,          83  01         05
     [4] IMPLICIT  SEQUENCE                                   A4 16
        {
        [0] IMPLICIT proposedVersionNumber,                   80 01          01
        [1] IMPLICIT proposedParameterCBB,                    81 03          05 f8 00
        [2] IMPLICIT servicesSupportedCalling                 82 0c          03 ee 19 00 18
                                                                             00 02 00 00 00
                                                                             FD 18
        }
    }
 }
```

where :	localDetailCalling (maxProposedMMSPduSize) ::=	2048 bytes
	proposedMaxServOutstandingCalling ::=	5
	proposedMaxServOutstandingCalled ::=	5
	proposedDataStructureNestingLevel (NEST) ::=	5
	proposedVersionNumber ::=	1 (MMS IS)
	proposedParameterCBB ::=	

note: 81 03 05 f8 00 indicates BITSTRING of length 3 bytes, the 05 indicates number of unused bits

str1	(bit 0 / array support / MSB of F8)	supported
str2	(bit 1 / structure support)	supported
vnam	(bit 2 / named variable support)	supported
valt	(bit 3 /alternate access support)	supported
vadr	(bit 4/ unnamed variable support)	supported
viscera	(bit 5/ scattered access support)	not-supported
toy	(bit 6/ third party operations support)	not-supported
villas	(bit 7/ named variable list support)	not-supported
real	(bit 8 / ASN.1 real data type support)	not-supported
ache	(bit 9/ acknowledge event condition support)	not-supported
chi	(bit 10 / condition event support)	not-supported

servicesSupportedCalling ::= see ISO/IEC-9506

Initiate-Response

MMSPdu Received ::=

```
A9 25 80 02 08 00 81 01  05 82 01 05 83 01 05 A4
16 80 01 01 81 03 05 F8  00 82 0C 03 EE 19 00 18
00 02 00 00 00 FD 18
```

```
initiate-ResponsePDU ::=                                      TAG/Length      VALUE
 {
  [9] IMPLICIT SEQUENCE                                       A9 25
    {
     [0] IMPLICIT localDetailCalled,                          80 02           08 00
     [1] IMPLICIT negotiatedMaxServOutstandingCalling,        81 01           05
     [2] IMPLICIT negotiatedMaxServOutstandingCalled,         82 01           05
     [3] IMPLICIT negotiatedDataStructureNestingLevel,        83  01          05
     [4] IMPLICIT  SEQUENCE                                   A4 16
        {
         [0] IMPLICIT negotiatedVersionNumber,                80 01           01
         [1] IMPLICIT negotiatedParameterCBB,                 81 03           05 f8 00
         [2] IMPLICIT servicesSupportedCalled                 82 0c           03 ee 19 00 18
                                                                              00 02 00 00 00
                                                                              FD 18
        }
     }
  }
```

where :   localDetailCalled (maxProposedMMSPduSize) ::=   2048 bytes
negotiatedMaxServOutstandingCalling ::=   5
negotiatedMaxServOutstandingCalled ::=   5
negotiatedDataStructureNestingLevel (NEST) ::=   5
negotiatedVersionNumber ::=   1 (MMS IS)
negotiatedParameterCBB ::=

note: 81 03 05 f8 00 indicates BITSTRING of length 3 bytes, the 05 indicates number of unused bits

str1	(bit 0 / array support / MSB of F8)	supported
str2	(bit 1 / structure support)	supported
vnam	(bit 2 / named variable support)	supported
valt	(bit 3 /alternate access support)	supported
vadr	(bit 4/ unnamed variable support)	supported
vsca	(bit 5/ scattered access support)	not-supported
tpy	(bit 6/ third party operations support)	not-supported
vlis	(bit 7/ named variable list support)	not-supported
real	(bit 8 / ASN.1 real data type support)	not-supported
akec	(bit 9/ acknowledge event condition support)	not-supported
cei	(bit 10 / condition event support)	not-supported

servicesSupportedCalling ::= see ISO/IEC-9506

Conclude-

MMSPdu Received ::=

8B 00

```
Conclude-PDU::=
  {
  [11] IMPLICIT NULL                                8B 00
  }
```

Conclude-Response

MMSPdu Received ::=

8C 00

```
Conclude-ResponsePDU::=
  {
  [12] IMPLICIT NULL                                8C 00
  }
```

VMD Management

Identify-

MMSPdu Received ::=

A0 05 02 01 01 82 00

Identify-PDU ::=	**TAG/Length**	**Value**
{		
[0] IMPLICIT SEQUENCE	A0 05	
{		
invokeID,	02 01	01
[2] IMPLICIT NULL	82 00	
}		
}		

where: invokeID ::= 01

Identify-Response

MMSPdu Received ::=

```
A1 2A 02 01 01 A2 25 80  0B 53 49 53 43 4F 2C 20
49 6E 63 2E 81 10 41 58  53 34 2D 4D 4D 53 2D 31
33 32 2D 30 31 38 82 04  32 2E 30 30
```

```
Identify-ResponsePDU ::=                              TAG/Length     Value
 {
 [1] IMPLICIT SEQUENCE                                A1 2A
    {
    invokeID,                                         02 01          01
    [2] IMPLICIT SEQUENCE                             A2 25
       {
       [0] IMPLICIT vendorName,                       80 0B          53 49 53 43 4F
                                                                     2C 20 49 6E 63
                                                                      2E
       [1] IMPLICIT modelName,                        81 10          41 58 53 34 2D
                                                                     4D 4D 53 2D 31
                                                                      33 32 2D 30 31
                                                                     38
       [2] IMPLICIT revision                          82 04          32 2E 30 30
       }
    }
 }
```

where: invokeID::= 01

note: matching of response to is done by matching invokeID with response invokeID.

vendorName::= "SISCO, Inc"

modelName::= "AXS4-MMS-132-018"

revision::= "2.00"

Variable Management

Read-

MMSPdu Received ::=

A0 1E 02 01 0A A4 19 A1 17 A0 15 30 13 A0 11 80
0F 66 65 65 64 65 72 31 5F 33 5F 70 68 61 73 65

Read-PDU ::=	**TAG/Length**	**Value**
{		
[0] IMPLICIT SEQUENCE	A0 1E	
{		
invokeID,	02 01	0A
[4] IMPLICIT SEQUENCE	A4 19	
{		
[1]EXPLICIT SEQUENCE	A1 17	
{		
[0] IMPLICIT SEQUENCE OF SEQUENCE	30 13	
{		
[0]EXPLICIT SEQUENCE	A0 11	
{		
[0] IMPLICIT Identifier	80 0F	66 65 65 64 65 72 31 5f 33 5f 70 68 61 73 65
}		
}		
}		
}		
}		
}		
}		

where: invokeID::= 0A
Identifier (name of variable to read) "feeder1_3_phase"

The following is the same PDU using PER:

```
Read-PDU ::=                                    TAG/Length    Value
  {
  [0]  IMPLICIT SEQUENCE                         00
     {
                                                 0        - Option Bitmap
                                                            ListOfModifier not present

      invokeID,                                  01                0A – length 1
      [4] IMPLICIT SEQUENCE                      4
         {
         [1]EXPLICIT SEQUENCE                    0          - Option Bitmap
                                                             specificationResult not present

                                                 1
           {
           [0] IMPLICIT SEQUENCE OF SEQUENCE     0

             {
             [0]EXPLICIT SEQUENCE                0
               {
               [0] IMPLICIT Identifier           03  0F            66 65 65 64 65 72 31 5f
                                                                   33 5f 70 68 61 73 65
               }
             }
           }
         }
       }
     }
   }
```

where: invokeID::= 0A
Identifier (name of variable to read) "feeder1_3_phase"

Read-ResponsePDU

The Read-Response, used in this example, is a reply to a Read-PDU for the Named Variable "feeder1_3_phase". The actual data of the variable is a structure consisting of two(2) INTEGER values. In 'c' notation:

```
typedef struct var_def
        {
        int     a;
        int     b;
        } VAR_DEF;

VAR_DEF feeder1_3_phase;
```

In order to make the decoded response more legible, the MMS Data Production will be shown in advance of the actual example:

```
Data ::= CHOICE {
        [1] IMPLICIT SEQUENCE OF,     -- arrayed data
        [2] IMPLICIT SEQUENCE OF,     -- structured data
        [3] IMPLICIT BOOLEAN,
        [4] IMPLICIT  BIT STRING,
        [5] IMPLICIT INTEGER,         -- signed int
        [6] IMPLICIT INTEGER,         -- unsigned int
        [7] IMPLICIT FloatingPoint,
        [9] IMPLICIT OCTET STRING,
        [10] IMPLICIT VisibleString,
        [11] IMPLICIT GeneralizedTime,
        [12] IMPLICIT TimeofDay,
        [13] IMPLICIT INTEGER,                  -- BCD
        [14] IMPLICIT BIT STRING,     -- boolean array
        [15] IMPLICIT OBJECT IDENTIFIER
                }
```


The encoded structure of the encoded data can be determined via VAR_DEF.

```
VAR_DEF::=                    TAG
    struct {                  A2
        int     a;            85
        int     b;            85
        }
```

MMSPdu Received ::=

```
A1 0F 02 01 0A A4 0A A1  08 A2 06 85 01 00 85 01
00
```

```
Read-ResponsePDU::=                              TAG/Length    Value
  {
  [1] SEQUENCE                                   A1 0F
    {
    invokeID,                                    02 01         0A
    [4] SEQUENCE                                 A4 0A
      {
      [1] IMPLICIT SEQUENCE OF                   A1 08         -- start of accessResult(s)
        {
        Data of feeder1_3_phase
              struct {                           A2 06
                    int      a;                  85 01         00
                    int      b;                  85 01         00
                    }
        }
      }
    }
  }
```

```
where:  invokeID::=           0A
        value of a::=         00
        value of b::=         00
```


The following is the same PDU encoded in PER.

Read-ResponsePDU::=	TAG/Length	Value
{		
[1] SEQUENCE	01	
{		
invokeID,	01	0A
[4] SEQUENCE	04	
	0	- Option Bitmask AccessSpecification not present
{		
[1] IMPLICIT SEQUENCE OF	01	-- start of accessResult(s)
{		
Data of feeder1_3_phase		
struct {	2	
int a;	51	00
int b;	51	00
}		
}		
}		
}		
}		

where: invokeID::= 0A
value of a::= 00
value of b::= 00

Practice PDUs

In order to allow the reader to practice cracking the MMS-PDUs, the following traces are being provided.

```
A0 0E 02 01 0A A1 09 A0  03 80 01 00 A1 02 80 00


A1 67 02 01 0A A1 62 A0  5D 1A 0B 54 65 6D 70 65
72 61 74 75 72 65 1A 0C  54 65 6D 70 65 72 61 74
75 72 65 31 1A 07 61 72  72 61 79 5F 35 1A 04 62
6F 6F 6C 1A 0F 66 65 65  64 65 72 31 5F 33 5F 70
68 61 73 65 1A 05 66 6C  6F 61 74 1A 0F 68 65 72
62 73 5F 74 65 73 74 5F  74 79 70 65 1A 08 75 6E
73 69 67 6E 65 64 81 01  00

--
A0 18 02 01 0B A6 13 A0  11 80 0F 66 65 65 64 65
72 31 5F 33 5F 70 68 61  73 65

A1 34 02 01 0B A6 2F 80  01 00 A1 16 81 14 66 65
65 64 65 72 31 5F 33 5F  70 68 61 73 65 24 41 64
64 72 A2 12 A2 10 A1 0E  30 05 A1 03 85 01 10 30
05 A1 03 85 01 10

--

A0 1E 02 01 0C A4 19 A1  17 A0 15 30 13 A0 11 80
0F 66 65 65 64 65 72 31  5F 33 5F 70 68 61 73 65

A1 0F 02 01 0C A4 0A A1  08 A2 06 85 01 01 85 01
02

--
A0 26 02 01 0D A5 21 A0  15 30 13 A0 11 80 0F 66
65 65 64 65 72 31 5F 33  5F 70 68 61 73 65 A0 08
A2 06 85 01 00 85 01 17

A1 07 02 01 0D A5 02 81  00

--
```


Cheat Sheets

The following tables are repeated from the body of the discussion to use as “cheat sheets” in reviewing packets on a non-ASN.1/MMS aware communications monitor:

Bits 7,6	Bit 5	Description, key words
0 0	0	Description: Universal Tag, Primitive Example keywords: INTEGER, BITSTRING, BOOLEAN
0 0	1	Description: Universal Tag, Constructed Example keyword: SEQUENCE, SEQUENCE OF
1 0	0	Description: Context Specific, Primitive Example Keyword: IMPLICIT
1 0	1	Description: Context Specific, Constructed Example Keywords: IMPLICIT SEQUENCE IMPLICIT SEQUENCE OF

Keyword	Tag Value (hex)
BOOLEAN	01
INTEGER	02
BITSTRING	03
OCTETSTRING	04
NULL	05
OBJECT IDENTIFIER	06
SEQUENCE	10
IA5STRING	16
UTCTIME	17
GENERALIZETIME	18
VISIBLESTRING	1A

Common MMS Data Value Encoding Examples			
Data Type	MMS Tag (hex)	Example Value	Encoding
array	81	int [2] = {0,1]	81 06 85 01 00 85 01 01 ^ ^ ^ | | +- 1 | +----------- 0 +------------------- array

Common MMS Data Value Encoding Examples			
Data Type	MMS Tag (hex)	Example Value	Encoding

structure	82	struct {
		int 0,
		bool TRUE
		}
			82 06 85 01 00 83 01 01
			^ ^ ^
			| | + TRUE
			| +------------ 0
			+------------------- struct

boolean	83	TRUE	83 01 01
bit-string	84	1010 (bin)	84 02 04 A0
integer	85	255 -255	85 02 00 ff 85 02 80 ff
unsigned	86	255	86 01 ff
floating-point	87	1.0	87 05 08 3f 80 00 00 -- IEEE Format
octet-string	89	01 02 (hex)	89 02 01 02
visible-string	8a	"ab"	8a 02 61 62
timeofday	8c	1:00:05 1:00:05, 12/31/96	8c 04 .. .. 8c 06 .. ..
bcd	8d	09 09 09 (hex)	8d 02 03 e7 -- Integer value 999
booleanArray	8e	1010	8e 03 04 a0

MMS PDU	TAG
confirmed-PDU	a0
confirmed-ResponsePDU	a1
confirmed-ErrorPDU	a2
unconfirmed-PDU	a3
rejectPDU	a4
cancel-PDU	a5
cancel-ResponsePDU	a6
cancel-ErrorPDU	a7
initiate-PDU	a8
initiate-ResponsePDU	a9
initiate-ErrorPDU	aa
conclude-PDU	ab
conclude-ResponsePDU	ac
conclude-ErrorPDU	ad

Data Access Errors	Error
object-invalidated	0
hardware-fault	1
temporarily-unavailable	2
object-access-denied	3
object-undefined	4
invalid-address	5
type-unsupported	6
type-inconsistent	7
object-attribute-inconsistent	8
object-access-unsupported	9
object-non-existent	a

MMS Confirmed Services TAG Table			
	TAG	Encoded TAG	
MMS Service	(dec)	Request	Response
status	0	80	a0
getNameList	1	a1	a1
identify	2	82	a2
rename	3	a3	83
read	4	a4	a4
write	5	a5	a5
getVariableAccessAttributes	6	a6	a6
defineNamedVariable	7	a7	87
defineScatteredAccess	8	a8	88
getScatteredAccessAttributes	9	a9	a9
deleteVariableAccess	10	aa	aa
defineNamedVariableList	11	ab	8b
getNamedVariableListAttributes	12	ac	ac
deleteNamedVariableList	13	ad	ad
defineNamedType	14	ae	8e
getNamedTypeAttributes	15	af	af
deleteNamedType	16	b0	b0
input	17	b1	91
output	18	b2	92
takeControl	19	b3	b3
relinquishControl	20	b4	94
defineSemaphore	21	b5	95
deleteSemaphore	22	b6	96
reportSemaphoreStatus	23	b7	b7
reportPoolSemaphoreStatus	24	b8	b8
reportSemaphoreEntryStatus	25	b9	b9
initiateDownloadSequence	26	ba	9a
downloadSegment	27	9b	bb
terminateDownloadSequence	28	bc	9c
initiateUploadSequence	29	9d	bd
uploadSegment	30	9e	be
terminateUploadSequence	31	9f 1f	9f 1f
DomainDownload	32	bf 20	9f 20
DomainUpload	33	bf 21	9f 21
loadDomainContent	34	bf 22	9f 22
storeDomainContent	35	bf 23	9f 23
deleteDomain	36	9f 24	9f 24
getDomainAttributes	37	9f 25	bf 25
createProgramInvocation	38	bf 26	9f 26
deleteProgramInvocation	39	9f 27	9f 27
start	40	bf 28	9f 28
stop	41	bf 29	9f 29
resume	42	bf 2a	9f 2a
reset	43	bf 2b	9f 2b

MMS Confirmed Services TAG Table			
	TAG	Encoded TAG	
MMS Service	(dec)	Request	Response
kill	44	bf 2c	9f 2c
getProgramInvocationAttributes	45	9f 2d	bf 2d
obtainFile	46	bf 2e	9f 2e
defineEventCondition	47	bf 2f	9f 2f
deleteEventCondition	48	bf 30	9f 30
getEventConditionAttributes	49	bf 31	bf 31
reportEventConditionStatus	50	bf 32	bf 32
alterEventConditionMonitoring	51	bf 33	9f 33
triggerEvent	52	bf 34	9f 34
defineEventAction	53	bf 35	9f 35
deleteEventAction	54	bf 36	9f 36
getEventActionAttributes	55	bf 37	bf 37
reportEventActionStatus	56	bf 38	9f 38
defineEventEnrollment	57	bf 39	9f 39
deleteEventEnrollment	58	bf 3a	9f 3a
alterEventEnrollment	59	bf 3b	bf 3b
reportEventEnrollmentStatus	60	bf 3c	bf 3c
getEventEnrollmentAttributes	61	bf 3d	bf 3d
acknowledgeEventNotification	62	bf 3e	9f 3e
getAlarmSummary	63	bf 3f	bf 3f
getAlarmEnrollmentSummary	64	bf 40	bf 40
readJournal	65	bf 41	bf 41
writeJournal	66	bf 42	9f 42
initializeJournal	67	bf 43	9f 43
reportJournalStatus	68	bf 44	bf 44
createJournal	69	bf 45	9f 45
deleteJournal	70	bf 46	9f 46
getCapabilityList	71	bf 47	bf 47
fileOpen	72	bf 48	bf 48
fileRead	73	9f 49	bf 49
fileClose	74	9f 4a	9f 4a
fileRename	75	bf 4b	9f 4b
fileDelete	76	bf 4c	9f 4c
fileDirectory	77	bf 4d	bf 4d