17

ILMI

ATM Forum ILMI specification 4.0 1996-0
IETF RFC 115, May 1990; RFC 1213, March 1991; RFC 1157, May 1990
ATM Forum UNI 3.0; "Managing Internetworks with SNMP" by Mark A. Miller, M&T Books, 1993

The Simple Network Management Protocol (SNMP) and an ATM UNI Management Information Base (MIB) are required to provide any ATM user device with status and configuration information concerning the virtual path and channel connection available at its UNI. In addition, their global operations and network management information may facilitate diagnostic procedures at the UNI.

The Interim Local Management Interface (ILMI) provides bi-directional exchange of management information between UNI Management Entities (UMEs). Both UMEs must contain the same MIB, even though the semantics of some MIB objects may be interpreted differently. Many types of equipment use this ATM UNI ILMI, e.g., high layer switches, workstations, computers with ATM interface, ATM network switches and more.

MIB Names

The following is a list, in tree form, of MIB names.

```
enterprises
353 atmForum
        1 atmForumAdmin
            2 atmfTransmissionTypes
             1 atmfUnknownType
             2 atmfSonetSTS3c
             3 atmfDs3
             4 atmf4B5B
             5 atmf8B10B
            3 atmfMediaTypes
             1 atmfMediaUnknownType
             2 atmfMediaCoaxCable
             3 atmfMediaSingleMode
             4 atmfMediaMultiMode
             5 atmfMediaStp
             6 atmfMediaUtp
            4 atmTrafficDescrTypes
             1 atmfNoDescriptor
             2 atmfPeakRate
             3 atmfNoClpNoScr
             4 atmfClpNoTaggingNoScr
             5 atmfClpTaggingNoScr
             6 atmfNoClpScr
             7 atmfClpNoTaggingScr
             8 atmfClpTaggingScr
            5 atmfSrvcRegTypes
             1 atmfSrvcRegLecs
        2 atmForumUni
             1 atmfPhysicalGroup
                 1 atmfPortTable
                     1 atmfPortEntry
                         1 atmfPortIndex
                         2 atmfPortAddress
                         3 atmfPortTransmissionType
                         4 atmfPortMediaType
                         5 atmfPortOperStatus
            6 atmfPortSpecific
2 atmfAtmLayerGroup
    1 atmfAtmLayerTable
        1 atmfAtmLayerEntry
            1 atmfAtmLayerIndex
            2 atmfAtmLayerMaxVPCs
            3 atmfAtmLayerMaxVCCs
            4 atmfAtmLayerConfiguredVPCs
            5 atmfAtmLayerConfiguredVCCs
            6 atmfAtmLayerMaxVpiBits
            7 atmfAtmLayerMaxVciBits
            8 atmfAtmLayerUniType
3 atmfAtmStatsGroup
    1 atmfAtmStatsTable
        1 atmfAtmStatsEntry
            1 atmfAtmStatsIndex
            2 atmfAtmStatsReceivedCells
            3 atmfAtmStatsDroppedReceivedCells
            4 atmfAtmStatsTransmittedCells
4 atmfVpcGroup
    1 atmVpcTable
        1 atmVpcEntry
            1 atmVpcPortIndex
            2 atmfVpcVpi
            3 atmfVpcOperStatus
            4 atmfVpcTransmitTrafficDescriptorType
            5 atmfVpcTransmitTrafficDescriptorParam1
            6 atmfVpcTransmitTrafficDescriptorParam2
            7 atmfVpcTransmitTrafficDescriptorParam3
            8 atmfVpcTransmitTrafficDescriptorParam4
            9 atmfVpcTransmitTrafficDescriptorParam5
            10 atmfVpcReceiveTrafficDescriptorType
            11 atmfVpcReceiveTrafficDescriptorParam1
            12 atmfVpcReceiveTrafficDescriptorParam2
            13 atmfVpcReceiveTrafficDescriptorParam3
            14 atmfVpcReceiveTrafficDescriptorParam4
            15 atmfVpcReceiveTrafficDescriptorParam5
            16 atmfVpcQoSCategory
            17 atmfVpcTransmitQoSClass
            18 atmfVpcReceiveQoSClass
5 atmfVccGroup
```

```
        1 atmfVccTable
            1 atmfVccEntry
                1 atmVccPortIndex
                2 atmfVccVpi
                3 atmfVccVci
                4 atmfVccOperStatus
                5 atmfVccTransmitTrafficDescriptorType
                6 atmfVccTransmitTrafficDescriptorParam1
                7 atmfVccTransmitTrafficDescriptorParam2
                8 atmfVccTransmitTrafficDescriptorParam3
                9 atmfVccTransmitTrafficDescriptorParam4
                10 atmfVccTransmitTrafficDescriptorParam5
                11 atmfVccReceiveTrafficDescriptorType
                12 atmfVccReceiveTrafficDescriptorParam1
                13 atmfVccReceiveTrafficDescriptorParam2
                14 atmfVccReceiveTrafficDescriptorParam3
                15 atmfVccReceiveTrafficDescriptorParam4
                16 atmfVccReceiveTrafficDescriptorParam5
                17 atmfVccQoSCategory
                18 atmfVccTransmitQoSClass
                19 atmfVccReceiveQoSClass
    8 atmfSrvcRegistryGroup
        1 atmfSrvcRegTable
            1 atmfSrvcRegEntry
                1 atmfSrvcRegPort
                1 atmfSrvcRegServiceID
                1 atmfSrvcRegATMAddress
                1 atmfSrvcRegAddressIndex
```

SNMP

RFC 1157: http://www.cis.ohio-state.edu/htbin/rfc/rfc1157.html

ILMI uses SNMP, which is designed to be simple and has a very straightforward architecture. The SNMP message is divided into two sections: a version identifier plus community name and a PDU.

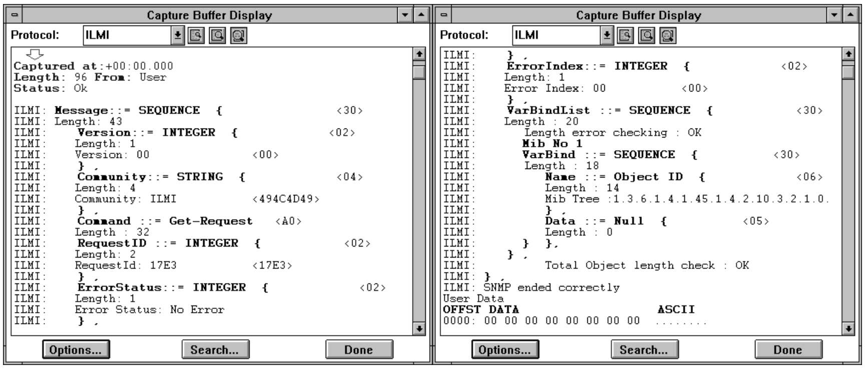
The version identifier and community name are sometimes referred to as the authentication header. The version number assures that both manager and agent are using the same version of SNMP. Messages between manager and agent containing different version numbers are discarded without further processing. The community name authenticates the manager before allowing access to the agent. The community name, along with the manager's IP address, is stored in the agent's community profile. If there is a difference between the manager and agent values for the community name, the agent will send an authentication failure trap message to the manager.

GetRequest and GetResponse PDUs

The manager uses the GetRequest PDU to retrieve the value of one or more object(s) from an agent. Under error-free conditions, the agent generates a GetResponse PDU. On both Request and Response PDUs there is a Request Index field that correlates the manager's request to the agent's response, an Error Status field which is set to noError, and an Error Index field which is set to zero. In this process, four errors are possible:

1. If a variable does not exactly match an available object, the agent returns a GetResponse PDU with the Error Status set to NoSuchName and the Error Index set the same as the index of the variable in question.
2. If a variable is of aggregate type, the Response is the same as above.
3. If the size of the appropriate GetResponse PDU would exceed a local limitation, the agent returns a GetResponse PDU of identical form, where the value of the Error Status is set to tooBig and the Error Index is set to 0.
4. If the value of a requested variable cannot be retrieved for any other reason, then the agent returns a GetResponse PDU, with the Error Status set to genErr and the Error Index set the same as the index of the variable in question.

The following is an example of a GetRequest PDU decode:



```
Captured at:+00:00.000
Length: 96 From: User
Status: Ok

ILMI: Message::= SEQUENCE  {                    <30>
ILMI: Length: 43
ILMI:    Version::= INTEGER  {                 <02>
ILMI:    Length: 1
ILMI:    Version: 00             <00>
ILMI:    } ,
ILMI:    Community::= STRING  {                <04>
ILMI:    Length: 4
ILMI:    Community: ILMI         <494C4D49>
ILMI:    } ,
ILMI:    Command ::= Get-Request   <A0>
ILMI:    Length : 32
ILMI:    RequestID ::= INTEGER  {               <02>
ILMI:    Length: 2
ILMI:    RequestId: 17E3         <17E3>
ILMI:    } ,
ILMI:    ErrorStatus::= INTEGER  {               <02>
ILMI:    Length: 1
ILMI:    Error Status: No Error
ILMI:    } ,
```

```
ILMI:    } ,
ILMI:    ErrorIndex::= INTEGER  {               <02>
ILMI:    Length: 1
ILMI:    Error Index: 00         <00>
ILMI:    } ,
ILMI:    VarBindList ::= SEQUENCE  {              <30>
ILMI:    Length : 20
ILMI:      Length error checking : OK
ILMI:      Mib No 1
ILMI:      VarBind ::= SEQUENCE  {            <30>
ILMI:      Length : 18
ILMI:         Name ::= Object ID  {              <06>
ILMI:         Length : 14
ILMI:         Mib Tree :1.3.6.1.4.1.45.1.4.2.10.3.2.1.0.
ILMI:         } ,
ILMI:         Data ::= Null  {           <05>
ILMI:         Length : 0
ILMI:      }  },
ILMI:    } ,
ILMI:         Total Object length check : OK
ILMI: } ,
ILMI: SNMP ended correctly
User Data
OFFST DATA                     ASCII
0000: 00 00 00 00 00 00 00 00  ........
```

GetRequest PDU

GetNextRequest PDU

The GetNextRequest PDU is used to retrieve one or more objects from an agent. Under error-free conditions, the agent generates a GetResponse PDU, with the same Request Index. The Variable Bindings contain the name and value associated with the lexicographic successor of each of the object identifiers (OIDs) noted in the GetNextRequest PDU. The main difference between GetRequest and GetNextRequest PDUs is that the GetNextRequest PDU retrieves the value of the next object within the agent's MIB view. Three possible errors may occur in this process:

1. If a variable in the Variable Bindings field does not lexicographically proceed the name of an object that may be retrieved, the agent returns a GetResponse with the Error Status set to noSuchName and the Error Index set to the same as the variable in question.

2. If the size of the appropriate GetResponse PDU would exceed a local limitation, the agent returns a GetResponse PDU of identical form, with the Error Status set to tooBig and the Error Index set to zero.

3. If the value of the lexicographic successor to a requested variable cannot be retrieved for any other reason, the agent returns the GetResponse PDU, with the Error Status set to genErr and the Error Index set the same as the index of the variable in question.

SetRequest PDU

The SetRequest PDU is used to assign a value to an object residing in the agent. When the agent receives the SetRequest PDU, it alters the values of the named objects to the values in the variable binding. Under error-free conditions, the agent generates a GetResponse PDU of identical form, except that the assigned PDU type is 2. Four different errors may occur in this process:

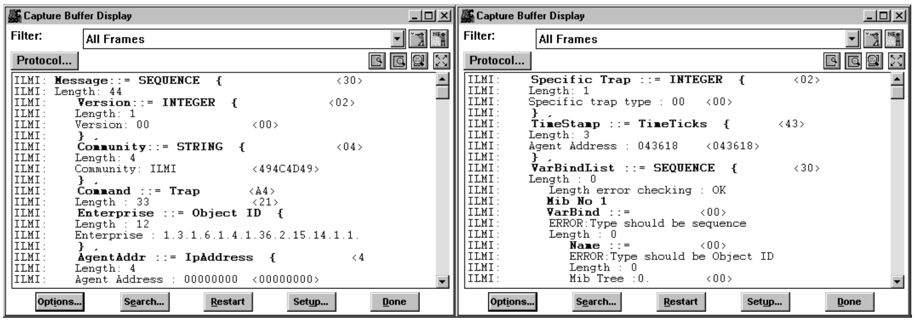
1. If a variable is not available for set operations within the relevant MIB view, the agent returns a GetResponse PDU with the Error Status set to NoSuchName (or readOnly) and the Error Index set the same as the index of the variable in question.
2. If a variable does not conform to the ASN.1 type, length and value, the agent returns a GetResponse with the Error Status set to badValue and the same Error Index.
3. If the size of the appropriate GetResponse PDU exceeds a local limitation, the agent returns a GetResponse PDU of identical form, with the Error Status set to tooBig and the Error Index set to zero.
4. If the value of a requested variable cannot be altered for any other reason, the agent returns a GetResponse PDU, with the Error Status set to genErr and the Error Index set the same as the index of the variable in question.

Trap PDU

The last PDU type is the Trap PDU which has a different format from the other four PDUs. It contains the following fields:

- Enterprise field, which identifies the management enterprise under whose registration authority the trap was defined.
- Generic trap type, which provides more specific information on the event being reported. There are seven unique values for this field: coldStart, warmStart, linkDown, linkUp, authenticationFailure, egpNeighborLoss, and enterpriseSpecific.
- Specific Trap Type field, which identifies the specific Trap.
- Timestamp field, which represents the amount of time elapsed between the last initialization of the agent and the generation of that Trap.
- Variable bindings.

The following is an example of the Trap decode:



```
ILMI: Message::= SEQUENCE  {                      <30>
ILMI: Length: 44
ILMI:     Version::= INTEGER  {                 <02>
ILMI:     Length: 1
ILMI:     Version: 00                 <00>
ILMI:     } ,
ILMI:     Community::= STRING  {                <04>
ILMI:     Length: 4
ILMI:     Community: ILMI             <494C4D49>
ILMI:     } ,
ILMI:     Command ::= Trap           <A4>
ILMI:     Length : 33                <21>
ILMI:     Enterprise ::= Object ID  {
ILMI:     Length : 12
ILMI:     Enterprise : 1.3.1.6.1.4.1.36.2.15.14.1.1.
ILMI:     } ,
ILMI:     AgentAddr ::= IpAddress  {              <4
ILMI:     Length: 4
ILMI:     Agent Address : 00000000  <00000000>
```

```
ILMI:     Specific Trap ::= INTEGER  {        <02>
ILMI:     Length: 1
ILMI:     Specific trap type : 00   <00>
ILMI:     } ,
ILMI:     TimeStamp ::= TimeTicks  {        <43>
ILMI:     Length: 3
ILMI:     Agent Address : 043618    <043618>
ILMI:     } ,
ILMI:     VarBindList ::= SEQUENCE  {        <30>
ILMI:     Length : 0
ILMI:        Length error checking : OK
ILMI:       Mib No 1
ILMI:       VarBind ::=             <00>
ILMI:        ERROR:Type should be sequence
ILMI:        Length : 0
ILMI:           Name ::=            <00>
ILMI:           ERROR:Type should be Object ID
ILMI:           Length : 0
ILMI:           Mib Tree :0.         <00>
```

Trap PDU

SMI

General

SMI (Structure of Management Information) is the standard used for defining the rules of managed object identification. The SMI organizes, names, and describes information so that logical access can occur. The SMI states that each managed object must have a name, a syntax , and an encoding. The name or OID uniquely identifies the object. The syntax defines the data type, such as an integer or a string of octets. The encoding describes how the information associated with the managed object is serialized for transmission between machines.

SMI defines the syntax that retrieves and communicates information, controls the way information is placed into logical groups, and the naming mechanism, known as the object identifiers, that identify each managed object. This can be extended to include MIBs, which store management information. Managed objects are accessed via an MIB. Objects in the MIB are defined using Abstract Syntax Notation One (ASN.1). Each type of object (termed an object type) has a name, a syntax, and an encoding. The name is represented uniquely as an OBJECT IDENTIFIER, which is an administratively assigned name. The syntax defines the abstract data structure corresponding to that object type. For example, the structure of a given object type might be an INTEGER or OCTET STRING. The encoding of an object type is simply how instances of that object type are represented using the object's type syntax.

Object Identifier

An object identifier is a sequence of integers which traverse a global tree. The tree consists of a root connected to a number of labeled nodes via edges. Each node may, in turn, have children of its own which are labeled. In this case we may term the node a subtree. This process may continue to an arbitrary level of depth.

The root node is unlabeled, but has at least three children directly under it; one node is administrated by the International Organization for Standardization, with label iso(1); another is administrated by the International Telegraph and Telephone Consultative Committee, with label ccitt(0); and the third is jointly administered by the ISO and CCITT, joint-iso-ccitt(2). Under the iso(1) node, the ISO has designated one subtree for use by other (inter)national organizations, org(3). Of the children nodes present, two have been assigned to the US National Institute of Standards and Technology. One of these subtrees has been transferred by the NIST to the US Department of Defense, dod(6). DoD will allocate a node to the Internet community, to be administered by the Internet Activities Board (IAB) as follows:

```
internet   OBJECT IDENTIFIER::={iso org(3) dod(6) 1}  ->  1.3.6.1
```

In this subtree four nodes are present:

```
directory        OBJECT IDENTIFIER::={ internet 1 }
mgmt             OBJECT IDENTIFIER::={ internet 2 }
experimental     OBJECT IDENTIFIER::={ internet 3 }
private          OBJECT IDENTIFIER::={ internet 4 }
```

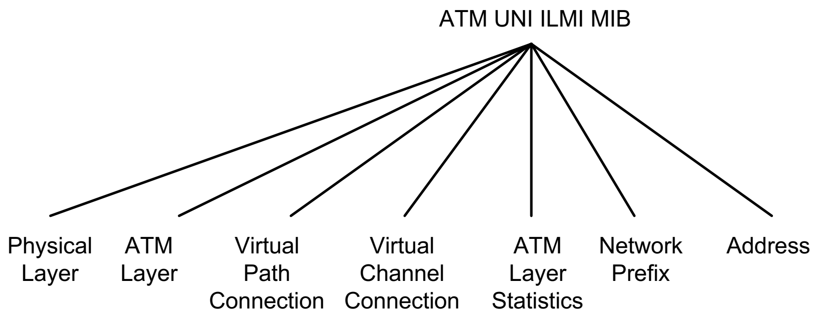
For example, the initial Internet standard MIB would be assigned management document number 1. -> { mgmt 1 } -> 1.3.6.1.2.1

The private(4) subtree is used to identify objects defined unilaterally. Administration of the private(4) subtree is delegated by the IAB to the Internet Assigned Numbers Authority for the Internet. Initially, this subtree has at least one child:

```
enterprises       OBJECT IDENTIFIER::={ private 1 }
```

The enterprises(1) subtree is used to permit parties providing networking subsystems to register models of their products.

Specific organizations have developed subtrees for private use for their products. One such tree is the ATM UNI MIB. Vendors can define private ATM UNI MIB extensions to support additional or proprietary features of their products. Objects in the MIB are defined using the subset of Abstract Syntax Notation One (ASN.1) defined by the SMI. The syntax of an object type defines the abstract data structure corresponding to that object type. The ASN.1 language is used for this purpose. The SMI purposely restricts the ASN.1 constructs which may be used. These restrictions are made for simplicity. The structure of ATM UNI ILMI MIB is illustrated in the following figure:



An entire tree group is either optional, conditionally required, or required. If a group is required, then every element in the group is required. If a group is conditionally required, every element in the group is required, if implemented.

Protocol Limitations

The following are some known SNMP limitations:

- ATM messages must be formatted according to SNMP version 1, not SNMP version 2.
- ALL SNMP messages will use the community name ILMI.
- In all SNMP Traps, the agent address field always has an IP Address value of 0.0.0.0.
- The supported traps are coldStart and enterpriseSpecific.
- In all SNMP traps, the timestamp field contains the value of the agent's sysUpTime MIB object at the time of trap generation. In all of the standard SNMP traps, the enterprise field in the Trap PDU contains the value of the agents sysObjectID MIB object.
- The size of messages can be up to 484 octets.