Network Working Group
Request for Comments: 2895
Obsoletes: 2074
Category: Standards Track

A. Bierman
C. Bucci
Cisco Systems, Inc.
R. Iddon
3Com, Inc.

August 2000

Remote Network Monitoring MIB Protocol Identifier Reference

Status of this Memo

Copyright Notice

Abstract

This memo defines a notation describing protocol layers in a protocol encapsulation, specifically for use in encoding INDEX values for the protocolDirTable, found in the RMON-2 MIB (Remote Network Monitoring Management Information Base) [RFC2021]. The definitions for the standard protocol directory base layer identifiers are also included.

The first version of the RMON Protocol Identifiers Document [RFC2074] has been split into a standards-track Reference portion (this document), and an Informational document. The RMON Protocol Identifier Macros document [RFC2896] now contains the non-normative portion of that specification.

This document obsoletes RFC 2074.

Table of Contents

1. The SNMP Network Management Framework

The SNMP Management Framework presently consists of five major components:

- An overall architecture, described in RFC 2571 [RFC2571].
- Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16, RFC 1155 [RFC1155], STD 16, RFC 1212 [RFC1212] and RFC 1215 [RFC1215]. The second version, called SMIv2, is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].
- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [RFC1157]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [RFC1901] and RFC 1906 [RFC1906]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [RFC1906], RFC 2572 [RFC2572] and RFC 2574 [RFC2574].
- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [RFC1157]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [RFC1905].
- A set of fundamental applications described in RFC 2573 [RFC2573] and the view-based access control mechanism described in RFC 2575 [RFC2575].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 2570 [RFC2570].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

This memo does not specify a MIB module.

2. Overview

The RMON-2 MIB [RFC2021] uses hierarchically formatted OCTET STRINGs to globally identify individual protocol encapsulations in the protocolDirTable.

This guide contains algorithms and the authoritative set of base layer protocol identifier macros, for use within INDEX values in the protocolDirTable.

This is the second revision of this document, and is intended to replace the first half of the first RMON-2 Protocol Identifiers document. [RFC2074].

2.1. Terms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Several terms are used throughout this document, as well as in the RMON-2 MIB [RFC2021], that should be introduced:

parent protocol:
: Also called 'parent'; The encapsulating protocol identifier for a specific protocol layer, e.g., IP is the parent protocol of UDP. Note that base layers cannot have parent protocols. This term may be used to refer to a specific encapsulating protocol, or it may be used generically to refer to any encapsulating protocol.

child protocol:
: Also called 'child'; An encapsulated protocol identifier for a specific protocol layer. e.g., UDP is a child protocol of IP. This term may be used to refer to a specific encapsulated protocol, or it may be used generically to refer to any encapsulated protocol.

layer-identifier:
: An octet string fragment representing a particular protocol encapsulation layer or sub-layer. A fragment consists of exactly four octets, encoded in network byte order. If present, child layer-identifiers for a protocol MUST have unique values among each other. (See section 3.3 for more details.)

protocol:
: A particular protocol layer, as specified by encoding rules in this document. Usually refers to a single layer in a given encapsulation. Note that this term is sometimes used in the RMON-2 MIB [RFC2021] to name a fully-specified protocol-identifier string. In such a case, the protocol-identifier string is named for its upper-most layer. A named protocol may also refer to any encapsulation of that protocol.

protocol-identifier string:
An octet string representing a particular protocol encapsulation, as specified by the encoding rules in this document. This string is identified in the RMON-2 MIB [RFC2021] as the protocolDirID object. A protocol-identifier string is composed of one or more layer-identifiers read from left to right. The left-most layer-identifier specifies a base layer encapsulation. Each layer-identifier to the right specifies a child layer protocol encapsulation.

protocol-identifier macro: Also called a PI macro; A macro-like textual construct used to describe a particular networking protocol. Only protocol attributes which are important for RMON use are documented. Note that the term 'macro' is historical, and PI macros are not real macros, nor are they ASN.1 macros. The current set of published RMON PI macros can be found in the RMON Protocol Identifier Macros document [RFC2896].

The PI macro serves several purposes:

- Names the protocol for use within the RMON-2 MIB [RFC2021].
- Describes how the protocol is encoded into an octet string.
- Describes how child protocols are identified (if applicable), and encoded into an octet string.
- Describes which protocolDirParameters are allowed for the protocol.
- Describes how the associated protocolDirType object is encoded for the protocol.
- Provides reference(s) to authoritative documentation for the protocol.

protocol-variant-identifier macro:
Also called a PI-variant macro; A special kind of PI macro, used to describe a particular protocol layer, which cannot be identified with a deterministic, and (usually) hierarchical structure, like most networking protocols.

Note that the PI-variant macro and the PI-macro are defined with a single set of syntax rules (see section 3.2), except that different sub-clauses are required for each type.

A protocol identified with a PI-variant macro is actually a variant of a well known encapsulation that may be present in the protocolDirTable. This is used to document the IANA assigned protocols, which are needed to identify protocols which cannot be practically identified by examination of 'appropriate network traffic' (e.g. the packets which carry them). All other protocols (which can be identified by examination of appropriate

network traffic) SHOULD be documented using the protocol-identifier macro. (See section 3.2 for details.)

protocol-parameter:
A single octet, corresponding to a specific layer-identifier in the protocol-identifier. This octet is a bit-mask indicating special functions or capabilities that this agent is providing for the corresponding protocol. (See section 3.2.6 for details.)

protocol-parameters string:
An octet string, which contains one protocol-parameter for each layer-identifier in the protocol-identifier. This string is identified in the RMON-2 MIB [RFC2021] as the protocolDirParameters object. (See the section 3.2.6 for details.)

protocolDirTable INDEX:
A protocol-identifier and protocol-parameters octet string pair that have been converted to an INDEX value, according to the encoding rules in section 7.7 of RFC 1902 [RFC1902].

pseudo-protocol:
A convention or algorithm used only within this document for the purpose of encoding protocol-identifier strings.

protocol encapsulation tree:
Protocol encapsulations can be organized into an inverted tree. The nodes of the root are the base encapsulations. The children nodes, if any, of a node in the tree are the encapsulations of child protocols.

2.2. Relationship to the Remote Network Monitoring MIB

This document is intended to identify the encoding rules for the OCTET STRING objects protocolDirID and protocolDirParameters. RMON-2 tables, such as those in the new Protocol Distribution, Host, and Matrix groups, use a local INTEGER INDEX (protocolDirLocalIndex) rather than complete protocolDirTable INDEX strings, to identify protocols for counting purposes. Only the protocolDirTable uses the protocolDirID and protocolDirParameters strings described in this document.

This document is intentionally separated from the RMON-2 MIB objects [RFC2021] to allow updates to this document without any republication of MIB objects.

This document does not discuss auto-discovery and auto-population of the protocolDirTable. This functionality is not explicitly defined by the RMON standard. An agent SHOULD populate the directory with the 'interesting' protocols on which the intended applications depend.

2.3. Relationship to the RMON Protocol Identifier Macros Document

The original RMON Protocol Identifiers document [RFC2074] contains the protocol directory reference material, as well as many examples of protocol identifier macros.

These macros have been moved to a separate document called the RMON Protocol Identifier Macros document [RFC2896]. This will allow the normative text (this document) to advance on the standards track with the RMON-2 MIB [RFC2021], while the collection of PI macros is maintained in an Informational RFC.

The PI Macros document is intentionally separated from this document to allow updates to the list of published PI macros without any republication of MIB objects or encoding rules. Protocol Identifier macros submitted from the RMON working group and community at large (to the RMONMIB WG mailing list at 'rmonmib@ietf.org') will be collected, screened by the RMONMIB working group, and (if approved) added to a subsequent version of the PI Macros document.

Macros submissions will be collected in the IANA's MIB files under the directory "ftp://ftp.isi.edu/mib/rmonmib/rmon2_pi_macros/" and in the RMONMIB working group mailing list message archive file www.ietf.org/mail-archive/working-groups/rmonmib/current/maillist.htm.

2.4. Relationship to the ATM-RMON MIB

The ATM Forum has standardized "Remote Monitoring MIB Extensions for ATM Networks" (ATM-RMON MIB) [AF-NM-TEST-0080.000], which provides RMON-like stats, host, matrix, and matrixTopN capability for NSAP address-based (ATM Adaption Layer 5, AAL-5) cell traffic.

2.4.1. Port Aggregation

It it possible to correlate ATM-RMON MIB data with packet-based RMON-2 [RFC2021] collections, but only if the ATM-RMON 'portSelGrpTable' and 'portSelTable' are configured to provide the same level of port aggregation as used in the packet-based collection. This will require an ATM-RMON 'portSelectGroup' to contain a single port, in the case of traditional RMON dataSources.

2.4.2. Encapsulation Mappings

The RMON PI document does not contain explicit PI macro support for "Multiprotocol Encapsulation over ATM Adaptation Layer 5" [RFC1483], or ATM Forum "LAN Emulation over ATM" (LANE) [AF-LANE-0021.000]. Instead, a probe must 'fit' the ATM encapsulation to one of the base layers defined in this document (i.e., llc, snap, or vsnap), regardless of how the raw data is obtained by the agent (e.g., VC-muxing vs. LLC-muxing, or routed vs. bridged formats). See section 3.2 for details on identifying and decoding a particular base layer.

An NMS can determine some of the omitted encapsulation details by examining the interface type (ifType) of the dataSource for a particular RMON collection:

RFC 1483 dataSource ifTypes:
- aal5(49)

LANE dataSource ifTypes:
- aflane8023(59)
- aflane8025(60)

These dataSources require implementation of the ifStackTable from the Interfaces MIB [RFC2233]. It is possible that some implementations will use dataSource values which indicate an ifType of 'atm(37)' (because the ifStackTable is not supported), however this is strongly discouraged by the RMONMIB WG.

2.4.3. Counting ATM Traffic in RMON-2 Collections

The RMON-2 Application Layer (AL) and Network Layer (NL) (host/matrix/topN) tables require that octet counters be incremented by the size of the particular frame, not by the size of the frame attributed to a given protocol.

Probe implementations must use the AAL-5 frame size (not the AAL-5 payload size or encapsulated MAC frame size) as the 'frame size' for the purpose of incrementing RMON-2 octet counters (e.g., 'nlHostInOctets', 'alHostOutOctets').

The RMONMIB WG has not addressed issues relating to packet capture of AAL-5 based traffic. Therefore, it is an implementation-specific matter whether padding octets (i.e., RFC 1483 VC-muxed, bridged 802.3 or 802.5 traffic, or LANE traffic) are represented in the RMON-1 'captureBufferPacketData' MIB object. Normally, the first octet of the captured frame is the first octet of the destination MAC address (DA).

2.5. Relationship to Other MIBs

The RMON Protocol Identifiers Reference document is intended for use with the protocolDirTable within the RMON MIB. It is not relevant to any other MIB, or intended for use with any other MIB.

3. Protocol Identifier Encoding

The protocolDirTable is indexed by two OCTET STRINGs, protocolDirID and protocolDirParameters. To encode the table index, each variable-length string is converted to an OBJECT IDENTIFIER fragment, according to the encoding rules in section 7.7 of RFC 1902 [RFC1902]. Then the index fragments are simply concatenated. (Refer to figures 1a - 1d below for more detail.)

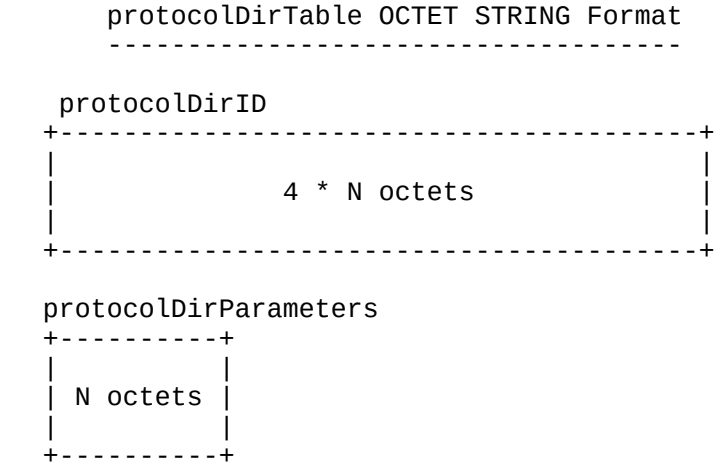
The first OCTET STRING (protocolDirID) is composed of one or more 4-octet "layer-identifiers". The entire string uniquely identifies a particular node in the protocol encapsulation tree. The second OCTET STRING, (protocolDirParameters) which contains a corresponding number of 1-octet protocol-specific parameters, one for each 4-octet layer-identifier in the first string.

A protocol layer is normally identified by a single 32-bit value. Each layer-identifier is encoded in the ProtocolDirID OCTET STRING INDEX as four sub-components [a.b.c.d], where ’a’ - ’d’ represent each byte of the 32-bit value in network byte order. If a particular protocol layer cannot be encoded into 32 bits, then it must be defined as an ’ianaAssigned’ protocol (see below for details on IANA assigned protocols).

The following figures show the differences between the OBJECT IDENTIFIER and OCTET STRING encoding of the protocol identifier string.

Fig. 1a
protocolDirTable INDEX Format

Fig. 1b
protocolDirTable OCTET STRING Format



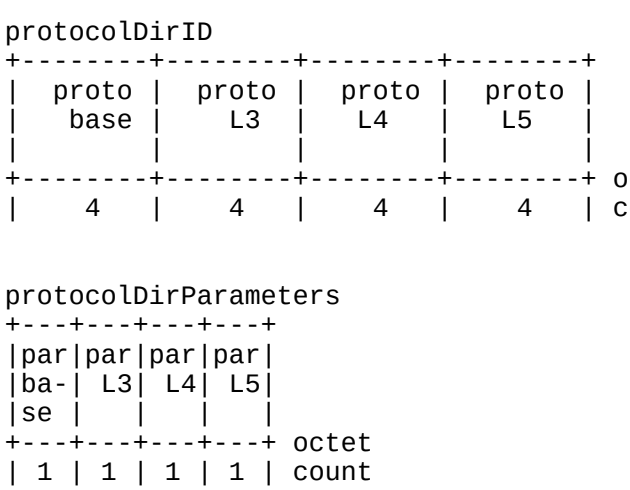
N is the number of protocol-layer-identifiers required for the entire encapsulation of the named protocol. Note that the layer following the base layer usually identifies a network layer protocol, but this is not always the case, (most notably for children of the 'vsnap' base-layer).

Fig. 1c
protocolDirTable INDEX Format Example

protocolDirID				protocolDirParameters						
c	proto	proto	proto	proto	c	par	par	par	par	
n	base	L(B+1)	L(B+2)	L(B+3)	n	ba-	L3	L4	L5	
t	(+flags)	L3	L4	L5	t	se				
1	4	4	4	4	1	1	1	1	1	subOID count

When encoded in a protocolDirTable INDEX, each of the two strings must be preceded by a length sub-component. In this example, N equals '4', the first 'cnt' field would contain the value '16', and the second 'cnt' field would contain the value '4'.

Fig. 1d
protocolDirTable OCTET STRING Format Example



Although this example indicates four encapsulated protocols, in practice, any non-zero number of layer-identifiers may be present, theoretically limited only by OBJECT IDENTIFIER length restrictions, as specified in section 3.5 of RFC 1902 [RFC1902].

3.1. ProtocolDirTable INDEX Format Examples

The following PI identifier fragments are examples of some fully encoded protocolDirTable INDEX values for various encapsulations.

```
 -- HTTP; fragments counted from IP and above
 ether2.ip.tcp.www-http =
    16.0.0.0.1.0.0.8.0.0.0.0.6.0.0.0.80.4.0.1.0.0

 -- SNMP over UDP/IP over SNAP
 snap.ip.udp.snmp =
    16.0.0.0.3.0.0.8.0.0.0.0.17.0.0.0.161.4.0.0.0.0

 -- SNMP over IPX over SNAP
 snap.ipx.snmp =
    12.0.0.0.3.0.0.129.55.0.0.144.15.3.0.0.0

 -- SNMP over IPX over raw8023
 ianaAssigned.ipxOverRaw8023.snmp =
    12.0.0.0.5.0.0.0.1.0.0.144.15.3.0.0.0
```

```
    -- IPX over LLC
    llc.ipx =
       8.0.0.0.2.0.0.0.224.2.0.0

    -- SNMP over UDP/IP over any link layer
    ether2.ip.udp.snmp
       16.1.0.0.1.0.0.8.0.0.0.0.17.0.0.0.161.4.0.0.0.0

    -- IP over any link layer; base encoding is IP over ether2
    ether2.ip
       8.1.0.0.1.0.0.8.0.2.0.0

    -- AppleTalk Phase 2 over ether2
    ether2.atalk
      8.0.0.0.1.0.0.128.155.2.0.0

    -- AppleTalk Phase 2 over vsnap
    vsnap.apple-oui.atalk
      12.0.0.0.4.0.8.0.7.0.0.128.155.3.0.0.0
```

3.2. Protocol Identifier Macro Format

The following example is meant to introduce the protocol-identifier macro. This macro-like construct is used to represent both protocols and protocol-variants.

If the 'VariantOfPart' component of the macro is present, then the macro represents a protocol-variant instead of a protocol. This clause is currently used only for IANA assigned protocols, enumerated under the 'ianaAssigned' base-layer. The VariantOfPart component MUST be present for IANA assigned protocols.

3.2.1. Lexical Conventions

The PI language defines the following keywords:

```
      ADDRESS-FORMAT
      ATTRIBUTES
      CHILDREN
      DECODING
      DESCRIPTION
      PARAMETERS
      PROTOCOL-IDENTIFIER
      REFERENCE
      VARIANT-OF
```

The PI language defines the following punctuation elements:

```
{     left curly brace
}     right curly brace
(     left parenthesis
)     right parenthesis
,     comma
::=   two colons and an equal sign
--    two dashes
```

3.2.2. Notation for Syntax Descriptions

An extended form of the BNF notation is used to specify the syntax of the PI language. The rules for this notation are shown below:

* Literal values are specified in quotes, for example "REFERENCE"

* Non-terminal items are surrounded by less than (<) and greater than (>) characters, for example <parmList>

* Terminal items are specified without surrounding quotes or less than and greater than characters, for example ’lcname’

* A vertical bar (|) is used to indicate a choice between items, for example ’number | hstr’

* Ellipsis are used to indicate that the previous item may be repeated one or more times, for example <parm>...

* Square brackets are used to enclose optional items, for example ["," <parm>]

* An equals character (=) is used to mean "defined as," for example ’<protoName> = pname’

3.2.3. Grammar for the PI Language

The following are "terminals" of the grammar and are identical to the same lexical elements from the MIB module language, except for hstr and pname:

```
<lc>     = "a" | "b" | "c" | ... | "z"
<uc>     = "A" | "B" | "C" | ... | "Z"
<letter> = <lc> | <uc>
<digit>  = "0" | "1" | ... | "9"
<hdigit> = <digit> | "a" | "A" | "b" | "B" | ... | "f" | "F"
```

```
<lcname> = <lc> [ <lcrest> ]
<lcrest> = ( <letter> | <digit> | "-" ) [ <lcrest> ]

<pname>  = ( <letter> | <digit> ) [ <pnrest> ]
<pnrest> = ( <letter> | <digit> | "-" | "_" | "*" ) [ <pnrest> ]

<number> = <digit> [ <number> ]  -- to a max dec. value of 4g-1

<hstr>   = "0x" <hrest>          -- to a max dec. value of 4g-1
<hrest>  = <hdigit> [ <hrest> ]

<lf>     = linefeed char
<cr>     = carriage return char
<eoln>   = <cr><lf> | <lf>

<sp>     = " "
<tab>    = "    "
<wspace> = { <sp> | <tab> | <eoln> } [<wspace>]

<string> = """ [ <strest> ] """
<strest> = ( <letter> | <digit> | <wspace> ) [ <strest> ]
```

The following is the extended BNF notation for the grammar with starting symbol <piFile>:

```
-- a file containing one or more Protocol Identifier (PI)
-- definitions
<piFile> = <piDefinition>...

-- a PI definition
<piDefinition> =
  <protoName> "PROTOCOL-IDENTIFIER"
      [ "VARIANT-OF" <protoName> ]
        "PARAMETERS" "{" [ <parmList> ] "}"
        "ATTRIBUTES" "{" [ <attrList> ] "}"
        "DESCRIPTION" string
      [ "CHILDREN" string ]
      [ "ADDRESS-FORMAT" string ]
      [ "DECODING" string ]
      [ "REFERENCE" string ]
        "::=" "{" <encapList> "}"

-- a protocol name
<protoName> = pname

-- a list of parameters
<parmList> = <parm> [ "," <parm> ]...
```

```
-- a parameter
<parm> = lcname [<wspace>] "(" [<wspace>]
          <nonNegNum> [<wspace>] ")" [<wspace>]

-- list of attributes
<attrList> = <attr> [ [<wspace>] "," [<wspace>] <attr> ]...

-- an attribute
<attr> = lcname [<wspace>] "(" [<wspace>]
          <nonNegNum> [<wspace>] ")"

-- a non-negative number
<nonNegNum> = number | hstr

-- list of encapsulation values
<encapList> = <encapValue> [ [<wspace>] ","
                [<wspace>] <encapValue> ]...

-- an encapsulation value
<encapValue> = <baseEncapValue> | <normalEncapValue>

-- base encapsulation value
<baseEncapValue> = <nonNegNum>

-- normal encapsulation value
 <normalEncapValue> = <protoName> <wspace> <nonNegNum>

-- comment
<two dashes> <text> <end-of-line>
```

3.2.4. Mapping of the Protocol Name

The "protoName" value, called the "protocol name" shall be an ASCII string consisting of one up to 64 characters from the following:

- "A" through "Z"
- "a" through "z"
- "0" through "9"
- dash (-)
- underbar (_)
- asterisk (*)
- plus(+)

The first character of the protocol name is limited to one of the following:

- "A" through "Z"
- "a" through "z"

"0" through "9"

This value SHOULD be the name or acronym identifying the protocol. Note that case is significant. The value selected for the protocol name SHOULD match the "most well-known" name or acronym for the indicated protocol. For example, the document indicated by the URL:

ftp://ftp.isi.edu/in-notes/iana/assignments/protocol-numbers

defines IP Protocol field values, so protocol-identifier macros for children of IP SHOULD be given names consistent with the protocol names found in this authoritative document. Likewise, children of UDP and TCP SHOULD be given names consistent with the port number name assignments found in:

ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers

When the "well-known name" contains characters not allowed in protocol names, they MUST be changed to a dash character ("-") . In the event that the first character must be changed, the protocol name is prepended with the letter "p", so the former first letter may be changed to a dash.

For example, z39.50 becomes z39-50 and 914c/g becomes 914c-g. The following protocol names are legal:

ftp, ftp-data, whois++, sql*net, 3com-tsmux, ocs_cmu

Note that it is possible in actual implementation that different encapsulations of the same protocol (which are represented by different entries in the protocolDirTable) will be assigned the same protocol name. The protocolDirID INDEX value defines a particular protocol, not the protocol name string.

3.2.5. Mapping of the VARIANT-OF Clause

This clause is present for IANA assigned protocols only. It identifies the protocol-identifier macro that most closely represents this particular protocol, and is known as the "reference protocol". A protocol-identifier macro MUST exist for the reference protocol. When this clause is present in a protocol-identifier macro, the macro is called a ’protocol-variant-identifier’.

Any clause (e.g. CHILDREN, ADDRESS-FORMAT) in the reference protocol-identifier macro SHOULD NOT be duplicated in the protocol-variant-identifier macro, if the ’variant’ protocols’ semantics are identical for a given clause.

Since the PARAMETERS and ATTRIBUTES clauses MUST be present in a protocol-identifier, an empty 'ParamList' and 'AttrList' (i.e. "PARAMETERS {}") MUST be present in a protocol-variant-identifier macro, and the 'ParamList' and 'AttrList' found in the reference protocol-identifier macro examined instead.

Note that if an 'ianaAssigned' protocol is defined that is not a variant of any other documented protocol, then the protocol-identifier macro SHOULD be used instead of the protocol-variant-identifier version of the macro.

3.2.6. Mapping of the PARAMETERS Clause

The protocolDirParameters object provides an NMS the ability to turn on and off expensive probe resources. An agent may support a given parameter all the time, not at all, or subject to current resource load.

The PARAMETERS clause is a list of bit definitions which can be directly encoded into the associated ProtocolDirParameters octet in network byte order. Zero or more bit definitions may be present. Only bits 0-7 are valid encoding values. This clause defines the entire BIT set allowed for a given protocol. A conforming agent may choose to implement a subset of zero or more of these PARAMETERS.

By convention, the following common bit definitions are used by different protocols. These bit positions MUST NOT be used for other parameters. They MUST be reserved if not used by a given protocol.

Bits are encoded in a single octet. Bit 0 is the high order (left-most) bit in the octet, and bit 7 is the low order (right-most) bit in the first octet. Reserved bits and unspecified bits in the octet are set to zero.

Table 3.1 Reserved PARAMETERS Bits

Bit	Name	Description
0	countsFragments	higher-layer protocols encapsulated within this protocol will be counted correctly even if this protocol fragments the upper layers into multiple packets.
1	tracksSessions	correctly attributes all packets of a protocol which starts sessions on well known ports or sockets and then transfers them to dynamically assigned ports or sockets thereafter (e.g. TFTP).

The PARAMETERS clause MUST be present in all protocol-identifier macro declarations, but may be equal to zero (empty).

3.2.6.1. Mapping of the 'countsFragments(0)' BIT

This bit indicates whether the probe is correctly attributing all fragmented packets of the specified protocol, even if individual frames carrying this protocol cannot be identified as such. Note that the probe is not required to actually present any re-assembled datagrams (for address-analysis, filtering, or any other purpose) to the NMS.

This bit MUST only be set in a protocolDirParameters octet which corresponds to a protocol that supports fragmentation and reassembly in some form. Note that TCP packets are not considered 'fragmented-streams' and so TCP is not eligible.

This bit MAY be set in more than one protocolDirParameters octet within a protocolDirTable INDEX, in the event an agent can count fragments at more than one protocol layer.

3.2.6.2. Mapping of the 'tracksSessions(1)' BIT

The 'tracksSessions(1)' bit indicates whether frames which are part of remapped sessions (e.g. TFTP download sessions) are correctly counted by the probe. For such a protocol, the probe must usually analyze all packets received on the indicated interface, and maintain some state information, (e.g. the remapped UDP port number for TFTP).

The semantics of the 'tracksSessions' parameter are independent of the other protocolDirParameters definitions, so this parameter MAY be combined with any other legal parameter configurations.

3.2.7. Mapping of the ATTRIBUTES Clause

The protocolDirType object provides an NMS with an indication of a probe's capabilities for decoding a given protocol, or the general attributes of the particular protocol.

The ATTRIBUTES clause is a list of bit definitions which are encoded into the associated instance of ProtocolDirType. The BIT definitions are specified in the SYNTAX clause of the protocolDirType MIB object.

Table 3.2 Reserved ATTRIBUTES Bits

Bit	Name	Description
0	hasChildren	indicates that there may be children of this protocol defined in the protocolDirTable (by either the agent or the manager).
1	addressRecognitionCapable	indicates that this protocol can be used to generate host and matrix table entries.

The ATTRIBUTES clause MUST be present in all protocol-identifier macro declarations, but MAY be empty.

3.2.8. Mapping of the DESCRIPTION Clause

The DESCRIPTION clause provides a textual description of the protocol identified by this macro. Notice that it SHOULD NOT contain details about items covered by the CHILDREN, ADDRESS-FORMAT, DECODING and REFERENCE clauses.

The DESCRIPTION clause MUST be present in all protocol-identifier macro declarations.

3.2.9. Mapping of the CHILDREN Clause

The CHILDREN clause provides a description of child protocols for protocols which support them. It has three sub-sections:

- Details on the field(s)/value(s) used to select the child protocol, and how that selection process is performed
- Details on how the value(s) are encoded in the protocol identifier octet string
- Details on how child protocols are named with respect to their parent protocol label(s)

The CHILDREN clause MUST be present in all protocol-identifier macro declarations in which the 'hasChildren(0)' BIT is set in the ATTRIBUTES clause.

3.2.10. Mapping of the ADDRESS-FORMAT Clause

The ADDRESS-FORMAT clause provides a description of the OCTET-STRING format(s) used when encoding addresses.

This clause MUST be present in all protocol-identifier macro declarations in which the 'addressRecognitionCapable(1)' BIT is set in the ATTRIBUTES clause.

3.2.11. Mapping of the DECODING Clause

The DECODING clause provides a description of the decoding procedure for the specified protocol. It contains useful decoding hints for the implementor, but SHOULD NOT over-replicate information in documents cited in the REFERENCE clause. It might contain a complete description of any decoding information required.

For 'extensible' protocols ('hasChildren(0)' BIT set) this includes offset and type information for the field(s) used for child selection as well as information on determining the start of the child protocol.

For 'addressRecognitionCapable' protocols this includes offset and type information for the field(s) used to generate addresses.

The DECODING clause is optional, and MAY be omitted if the REFERENCE clause contains pointers to decoding information for the specified protocol.

3.2.12. Mapping of the REFERENCE Clause

If a publicly available reference document exists for this protocol it SHOULD be listed here. Typically this will be a URL if possible; if not then it will be the name and address of the controlling body.

The CHILDREN, ADDRESS-FORMAT, and DECODING clauses SHOULD limit the amount of information which may currently be obtained from an authoritative document, such as the Assigned Numbers document [RFC1700]. Any duplication or paraphrasing of information should be brief and consistent with the authoritative document.

The REFERENCE clause is optional, but SHOULD be implemented if an authoritative reference exists for the protocol (especially for standard protocols).

3.3. Evaluating an Index of the ProtocolDirTable

The following evaluation is done after a protocolDirTable INDEX value has been converted into two OCTET STRINGs according to the INDEX encoding rules specified in the SMI [RFC1902].

Protocol-identifiers are evaluated left to right, starting with the protocolDirID, which length MUST be evenly divisible by four. The protocolDirParameters length MUST be exactly one quarter of the protocolDirID string length.

Protocol-identifier parsing starts with the base layer identifier, which MUST be present, and continues for one or more upper layer identifiers, until all OCTETs of the protocolDirID have been used. Layers MUST NOT be skipped, so identifiers such as 'SNMP over IP' or 'TCP over ether2' can not exist.

The base-layer-identifier also contains a 'special function identifier' which may apply to the rest of the protocol identifier.

Wild-carding at the base layer within a protocol encapsulation is the only supported special function at this time. (See section 4.1.1.2 for details.)

After the protocol-identifier string (which is the value of protocolDirID) has been parsed, each octet of the protocol-parameters string is evaluated, and applied to the corresponding protocol layer.

A protocol-identifier label MAY map to more than one value. For instance, 'ip' maps to 5 distinct values, one for each supported encapsulation. (see the 'IP' section under 'L3 Protocol Identifiers' in the RMON Protocol Identifier Macros document [RFC2896]).

It is important to note that these macros are conceptually expanded at implementation time, not at run time.

If all the macros are expanded completely by substituting all possible values of each label for each child protocol, a list of all possible protocol-identifiers is produced. So 'ip' would result in 5 distinct protocol-identifiers. Likewise each child of 'ip' would map to at least 5 protocol-identifiers, one for each encapsulation (e.g. ip over ether2, ip over LLC, etc.).

4. Base Layer Protocol Identifier Macros

The following PROTOCOL IDENTIFIER macros can be used to construct protocolDirID and protocolDirParameters strings.

An identifier is encoded by constructing the base-identifier, then adding one layer-identifier for each encapsulated protocol.

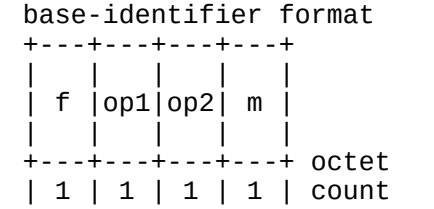
Refer to the RMON Protocol Identifier Macros document [RFC2896] for a listing of the non-base layer PI macros published by the working group. Note that other PI macro documents may exist, and it should be possible for an implementor to populate the protocolDirTable without the use of the PI Macro document [RFC2896].

4.1. Base Identifier Encoding

The first layer encapsulation is called the base identifier and it contains optional protocol-function information and the base layer (e.g. MAC layer) enumeration value used in this protocol identifier.

The base identifier is encoded as four octets as shown in figure 2.

Fig. 2
base-identifier format



The first octet ('f') is the special function code, found in table 4.1. The next two octets ('op1' and 'op2') are operands for the indicated function. If not used, an operand must be set to zero. The last octet, 'm', is the enumerated value for a particular base layer encapsulation, found in table 4.2. All four octets are encoded in network-byte-order.

4.1.1. Protocol Identifier Functions

The base layer identifier contains information about any special functions to perform during collections of this protocol, as well as the base layer encapsulation identifier.

The first three octets of the identifier contain the function code and two optional operands. The fourth octet contains the particular base layer encapsulation used in this protocol (fig. 2).

Table 4.1 Assigned Protocol Identifier Functions

Function	ID	Param1	Param2
none	0	not used (0)	not used (0)
wildcard	1	not used (0)	not used (0)

4.1.1.1. Function 0: None

If the function ID field (1st octet) is equal to zero, the 'op1' and 'op2' fields (2nd and 3rd octets) must also be equal to zero. This special value indicates that no functions are applied to the protocol identifier encoded in the remaining octets. The identifier represents a normal protocol encapsulation.

4.1.1.2. Function 1: Protocol Wildcard Function

The wildcard function (function-ID = 1), is used to aggregate counters, by using a single protocol value to indicate potentially many base layer encapsulations of a particular network layer protocol. A protocolDirEntry of this type will match any base-layer encapsulation of the same network layer protocol.

The 'op1' field (2nd octet) is not used and MUST be set to zero.

The 'op2' field (3rd octet) is not used and MUST be set to zero.

Each wildcard protocol identifier MUST be defined in terms of a 'base encapsulation'. This SHOULD be as 'standard' as possible for interoperability purposes. The lowest possible base layer value SHOULD be chosen. So, if an encapsulation over 'ether2' is permitted, than this should be used as the base encapsulation. If not then an encapsulation over LLC should be used, if permitted. And so on for each of the defined base layers.

It should be noted that an agent does not have to support the non-wildcard protocol identifier over the same base layer. For instance a token ring only device would not normally support IP over the ether2 base layer. Nevertheless it should use the ether2 base layer for defining the wildcard IP encapsulation. The agent MAY also support counting some or all of the individual encapsulations for the same protocols, in addition to wildcard counting. Note that the RMON-2 MIB [RFC2021] does not require that agents maintain counters for multiple encapsulations of the same protocol. It is an implementation-specific matter as to how an agent determines which protocol combinations to allow in the protocolDirTable at any given time.

4.2. Base Layer Protocol Identifiers

The base layer is mandatory, and defines the base encapsulation of the packet and any special functions for this identifier.

There are no suggested protocolDirParameters bits for the base layer.

The suggested value for the ProtocolDirDescr field for the base layer is given by the corresponding "Name" field in the table 4.2 below. However, implementations are only required to use the appropriate integer identifier values.

For most base layer protocols, the protocolDirType field should contain bits set for the 'hasChildren(0)' and 'addressRecognitionCapable(1)' attributes. However, the special 'ianaAssigned' base layer should have no parameter or attribute bits set.

By design, only 255 different base layer encapsulations are supported. There are five base encapsulation values defined at this time. Very few new base encapsulations (e.g. for new media types) are expected to be added over time.

Table 4.2 Base Layer Encoding Values

Name	ID
ether2	1
llc	2
snap	3
vsnap	4
ianaAssigned	5

```
 -- Ether2 Encapsulation

ether2 PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES {
     hasChildren(0),
        addressRecognitionCapable(1)
    }
    DESCRIPTION
       "DIX Ethernet, also called Ethernet-II."
    CHILDREN
       "The Ethernet-II type field is used to select child protocols.
       This is a 16-bit field.  Child protocols are deemed to start at
       the first octet after this type field.
```

```
       Children of this protocol are encoded as [ 0.0.0.1 ], the
       protocol identifier for 'ether2' followed by [ 0.0.a.b ] where
       'a' and 'b' are the network byte order encodings of the high
       order byte and low order byte of the Ethernet-II type value.

       For example, a protocolDirID-fragment value of:
          0.0.0.1.0.0.8.0 defines IP encapsulated in ether2.

       Children of ether2 are named as 'ether2' followed by the type
       field value in hexadecimal.  The above example would be declared
       as:
          ether2 0x0800"
    ADDRESS-FORMAT
       "Ethernet addresses are 6 octets in network order."
    DECODING
       "Only type values greater than 1500 decimal indicate Ethernet-II
       frames; lower values indicate 802.3 encapsulation (see below)."
    REFERENCE
       "The authoritative list of Ether Type values is identified by the
       URL:

          ftp://ftp.isi.edu/in-notes/iana/assignments/ethernet-numbers"
    ::= { 1 }

 -- LLC Encapsulation

llc PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES {
     hasChildren(0),
     addressRecognitionCapable(1)
    }
    DESCRIPTION
       "The Logical Link Control (LLC) 802.2 protocol."
    CHILDREN
       "The LLC Source Service Access Point (SSAP) and Destination
       Service Access Point (DSAP) are used to select child protocols.
       Each of these is one octet long, although the least significant
       bit is a control bit and should be masked out in most situations.
       Typically SSAP and DSAP (once masked) are the same for a given
       protocol - each end implicitly knows whether it is the server or
       client in a client/server protocol.  This is only a convention,
       however, and it is possible for them to be different.  The SSAP
       is matched against child protocols first.  If none is found then
       the DSAP is matched instead.  The child protocol is deemed to
       start at the first octet after the LLC control field(s).
```

```
       Children of 'llc' are encoded as [ 0.0.0.2 ], the protocol
       identifier component for LLC followed by [ 0.0.0.a ] where 'a' is
       the SAP value which maps to the child protocol.  For example, a
       protocolDirID-fragment value of:
          0.0.0.2.0.0.0.240

       defines NetBios over LLC.

       Children are named as 'llc' followed by the SAP value in
       hexadecimal.  So the above example would have been named:
          llc 0xf0"
    ADDRESS-FORMAT
       "The address consists of 6 octets of MAC address in network
       order.  Source routing bits should be stripped out of the address
       if present."
    DECODING
       "Notice that LLC has a variable length protocol header; there are
       always three octets (DSAP, SSAP, control).  Depending on the
       value of the control bits in the DSAP, SSAP and control fields
       there may be an additional octet of control information.

       LLC can be present on several different media.  For 802.3 and
       802.5 its presence is mandated (but see ether2 and raw 802.3
       encapsulations).  For 802.5 there is no other link layer
       protocol.

       Notice also that the raw802.3 link layer protocol may take
       precedence over this one in a protocol specific manner such that
       it may not be possible to utilize all LSAP values if raw802.3 is
       also present."
    REFERENCE
       "The authoritative list of LLC LSAP values is controlled by the
       IEEE Registration Authority:
       IEEE Registration Authority
          c/o Iris Ringel
          IEEE Standards Dept
          445 Hoes Lane, P.O. Box 1331
          Piscataway, NJ 08855-1331
          Phone +1 908 562 3813
          Fax: +1 908 562 1571"
    ::= { 2 }

 -- SNAP over LLC (Organizationally Unique Identifier, OUI=000)
 -- Encapsulation

snap PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES {
```

```
     hasChildren(0),
     addressRecognitionCapable(1)
    }
    DESCRIPTION
       "The Sub-Network Access Protocol (SNAP) is layered on top of LLC
       protocol, allowing Ethernet-II protocols to be run over a media
       restricted to LLC."
    CHILDREN
       "Children of 'snap' are identified by Ethernet-II type values;
       the SNAP Protocol Identifier field (PID) is used to select the
       appropriate child.  The entire SNAP protocol header is consumed;
       the child protocol is assumed to start at the next octet after
       the PID.

       Children of 'snap' are encoded as [ 0.0.0.3 ], the protocol
       identifier for 'snap', followed by [ 0.0.a.b ] where 'a' and 'b'
       are the high order byte and low order byte of the Ethernet-II
       type value.

       For example, a protocolDirID-fragment value of:
          0.0.0.3.0.0.8.0

       defines the IP/SNAP protocol.

       Children of this protocol are named 'snap' followed by the
       Ethernet-II type value in hexadecimal.  The above example would
       be named:

          snap 0x0800"
    ADDRESS-FORMAT
         "The address format for SNAP is the same as that for LLC"
    DECODING
       "SNAP is only present over LLC.  Both SSAP and DSAP will be 0xAA
       and a single control octet will be present.  There are then three
       octets of Organizationally Unique Identifier (OUI) and two octets
       of PID.  For this encapsulation the OUI must be 0x000000 (see
       'vsnap' below for non-zero OUIs)."
    REFERENCE
       "SNAP Identifier values are assigned by the IEEE Standards
       Office.  The address is:
            IEEE Registration Authority
            c/o Iris Ringel
            IEEE Standards Dept
            445 Hoes Lane, P.O. Box 1331
            Piscataway, NJ 08855-1331
            Phone +1 908 562 3813
            Fax: +1 908 562 1571"
    ::= { 3 }
```

```
 -- Vendor SNAP over LLC (OUI != 000) Encapsulation

vsnap PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES {
     hasChildren(0),
     addressRecognitionCapable(1)
    }
    DESCRIPTION
       "This pseudo-protocol handles all SNAP packets which do not have
       a zero OUI.  See 'snap' above for details of those that have a
       zero OUI value."
    CHILDREN
       "Children of 'vsnap' are selected by the 3 octet OUI; the PID is
       not parsed; child protocols are deemed to start with the first
       octet of the SNAP PID field, and continue to the end of the
       packet.  Children of 'vsnap' are encoded as [ 0.0.0.4 ], the
       protocol identifier for 'vsnap', followed by [ 0.a.b.c ] where
       'a', 'b' and 'c' are the 3 octets of the OUI field in network
       byte order.

       For example, a protocolDirID-fragment value of:
         0.0.0.4.0.8.0.7 defines the Apple-specific set of protocols
       over vsnap.

       Children are named as 'vsnap <OUI>', where the '<OUI>' field is
       represented as 3 octets in hexadecimal notation.

       So the above example would be named:
         'vsnap 0x080007'"
    ADDRESS-FORMAT
       "The LLC address format is inherited by 'vsnap'.  See the 'llc'
       protocol identifier for more details."
    DECODING
       "Same as for 'snap' except the OUI is non-zero and the SNAP
       Protocol Identifier is not parsed."
    REFERENCE
       "SNAP Identifier values are assigned by the IEEE Standards
       Office.  The address is:
            IEEE Registration Authority
            c/o Iris Ringel
            IEEE Standards Dept
            445 Hoes Lane, P.O. Box 1331
            Piscataway, NJ 08855-1331
            Phone +1 908 562 3813
            Fax: +1 908 562 1571"
    ::= { 4 }
```

```
 -- IANA Assigned Protocols

ianaAssigned PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES { }
    DESCRIPTION
       "This branch contains protocols which do not conform easily to
       the hierarchical format utilized in the other link layer
       branches.  Usually, such a protocol 'almost' conforms to a
       particular 'well-known' identifier format, but additional
       criteria are used (e.g. configuration-based), making protocol
       identification difficult or impossible by examination of
       appropriate network traffic (preventing the any 'well-known'
       protocol-identifier macro from being used).

       Sometimes well-known protocols are simply remapped to a different
       port number by one or more venders (e.g. SNMP). These protocols
       can be identified with the 'limited extensibility' feature of the
       protocolDirTable, and do not need special IANA assignments.

       A centrally located list of these enumerated protocols must be
       maintained by IANA to insure interoperability. (See section 2.3
       for details on the document update procedure.)  Support for new
       link-layers will be added explicitly, and only protocols which
       cannot possibly be represented in a better way will be considered
       as 'ianaAssigned' protocols.

       IANA protocols are identified by the base-layer-selector value [
       0.0.0.5 ], followed by the four octets [ 0.0.a.b ] of the integer
       value corresponding to the particular IANA protocol.

       Do not create children of this protocol unless you are sure that
       they cannot be handled by the more conventional link layers
       above."
    CHILDREN
       "Children of this protocol are identified by implementation-
       specific means, described (as best as possible) in the 'DECODING'
       clause within the protocol-variant-identifier macro for each
       enumerated protocol.

       Children of this protocol are encoded as [ 0.0.0.5 ], the
       protocol identifier for 'ianaAssigned', followed by [ 0.0.a.b ]
       where 'a', 'b' are the network byte order encodings of the high
       order byte and low order byte of the enumeration value for the
       particular IANA assigned protocol.
```

```
       For example, a protocolDirID-fragment value of:
          0.0.0.5.0.0.0.1

       defines the IPX protocol encapsulated directly in 802.3

       Children are named 'ianaAssigned' followed by the numeric value
       of the particular IANA assigned protocol.  The above example
       would be named:

          'ianaAssigned 1' "
    DECODING
       "The 'ianaAssigned' base layer is a pseudo-protocol and is not
       decoded."
    REFERENCE
       "Refer to individual PROTOCOL-IDENTIFIER macros for information
       on each child of the IANA assigned protocol."
    ::= { 5 }

 -- The following protocol-variant-identifier macro declarations are
 -- used to identify the RMONMIB IANA assigned protocols in a
 -- proprietary way, by simple enumeration.

ipxOverRaw8023 PROTOCOL-IDENTIFIER
    VARIANT-OF  ipx
    PARAMETERS      { }
    ATTRIBUTES  { }
    DESCRIPTION
       "This pseudo-protocol describes an encapsulation of IPX over
       802.3, without a type field.

       Refer to the macro for IPX for additional information about this
       protocol."
    DECODING
       "Whenever the 802.3 header indicates LLC a set of protocol
       specific tests needs to be applied to determine whether this is a
       'raw8023' packet or a true 802.2 packet.  The nature of these
       tests depends on the active child protocols for 'raw8023' and is
       beyond the scope of this document."
    ::= {
     ianaAssigned 1,             -- [0.0.0.1]
     802-1Q       0x05000001     -- 1Q_IANA [5.0.0.1]
    }
```

4.3. Encapsulation Layers

Encapsulation layers are positioned between the base layer and the network layer. It is an implementation-specific matter whether a probe exposes all such encapsulations in its RMON-2 Protocol Directory.

4.3.1. IEEE 802.1Q

RMON probes may encounter 'VLAN tagged' frames on monitored links. The IEEE Virtual LAN (VLAN) encapsulation standards [IEEE802.1Q] and [IEEE802.1D-1998], define an encapsulation layer inserted after the MAC layer and before the network layer. This section defines a PI macro which supports most (but not all) features of that encapsulation layer.

Most notably, the RMON PI macro '802-1Q' does not expose the Token Ring Encapsulation (TR-encaps) bit in the TCI portion of the VLAN header. It is an implementation specific matter whether an RMON probe converts LLC-Token Ring (LLC-TR) formatted frames to LLC-Native (LLC-N) format, for the purpose of RMON collection.

In order to support the Ethernet and LLC-N formats in the most efficient manner, and still maintain alignment with the RMON-2 ' collapsed' base layer approach (i.e., support for snap and vsnap), the children of 802dot1Q are encoded a little differently than the children of other base layer identifiers.

```
802-1Q   PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES {
     hasChildren(0)
    }
    DESCRIPTION
       "IEEE 802.1Q VLAN Encapsulation header.

       Note that the specific encoding of the TPID field is not
       explicitly identified by this PI macro.  Ethernet-encoded vs.
       SNAP-encoded TPID fields can be identified by the ifType of the
       data source for a particular RMON collection, since the SNAP-
       encoded format is used exclusively on Token Ring and FDDI media.
       Also, no information held in the TCI field (including the TR-
       encap bit) is identified in protocolDirID strings utilizing this
       PI macro."
```

CHILDREN

"The first byte of the 4-byte child identifier is used to distinguish the particular base encoding that follows the 802.1Q header. The remaining three bytes are used exactly as defined by the indicated base layer encoding.

In order to simplify the child encoding for the most common cases, the 'ether2' and 'snap' base layers are combined into a single identifier, with a value of zero. The other base layers are encoded with values taken from Table 4.2.

802-1Q Base ID Values

Base Layer	Table 4.2 Encoding	Base-ID Encoding
ether2	1	0
llc	2	2
snap	3	0
vsnap	4	4
ianaAssigned	5	5

The generic child layer-identifier format is shown below:

802-1Q Child Layer-Identifier Format

Base ID	base-specific format	
1	3	octet count

Base ID == 0

For payloads encoded with either the Ethernet or LLC/SNAP headers following the VLAN header, children of this protocol are identified exactly as described for the 'ether2' or 'snap' base layers.

Children are encoded as [0.0.129.0], the protocol identifier for '802-1Q' followed by [0.0.a.b] where 'a' and 'b' are the network byte order encodings of the high order byte and low order byte of the Ethernet-II type value.

For example, a protocolDirID-fragment value of:

0.0.0.1.0.0.129.0.0.0.8.0

defines IP, VLAN-encapsulated in ether2.

Children of this format are named as '802-1Q' followed by the type field value in hexadecimal.

So the above example would be declared as:
'802-1Q 0x0800'.

Base ID == 2

For payloads encoded with a (non-SNAP) LLC header following the VLAN header, children of this protocol are identified exactly as described for the 'llc' base layer.

Children are encoded as [0.0.129.0], the protocol identifier component for 802.1Q, followed by [2.0.0.a] where 'a' is the SAP value which maps to the child protocol. For example, a protocolDirID-fragment value of:
0.0.0.1.0.0.129.0.2.0.0.240

defines NetBios, VLAN-encapsulated over LLC.

Children are named as '802-1Q' followed by the SAP value in hexadecimal, with the leading octet set to the value 2.

So the above example would have been named:
'802-1Q 0x020000f0'

Base ID == 4

For payloads encoded with LLC/SNAP (non-zero OUI) headers following the VLAN header, children of this protocol are identified exactly as described for the 'vsnap' base layer.

Children are encoded as [0.0.129.0], the protocol identifier for '802-1Q', followed by [4.a.b.c] where 'a', 'b' and 'c' are the 3 octets of the OUI field in network byte order.

For example, a protocolDirID-fragment value of:
0.0.0.1.0.0.129.0.4.8.0.7 defines the Apple-specific set of protocols, VLAN-encapsulated over vsnap.

Children are named as '802-1Q' followed by the <OUI> value, which is represented as 3 octets in hexadecimal notation, with a leading octet set to the value 4.

So the above example would be named:
'802-1Q 0x04080007'.

```
      Base ID == 5
      ------------
      For payloads which can only be identified as 'ianaAssigned'
      protocols, children of this protocol are identified exactly as
      described for the 'ianaAssigned' base layer.

      Children are encoded as [ 0.0.129.0 ], the protocol identifier
      for '802-1Q', followed by [ 5.0.a.b ] where 'a' and 'b' are the
      network byte order encodings of the high order byte and low order
      byte of the enumeration value for the particular IANA assigned
      protocol.

      For example, a protocolDirID-fragment value of:
         0.0.0.1.0.0.129.0.5.0.0.0.1

      defines the IPX protocol, VLAN-encapsulated directly in 802.3

      Children are named '802-1Q' followed by the numeric value of the
      particular IANA assigned protocol, with a leading octet set to
      the value of 5.

      Children are named '802-1Q' followed by the hexadecimal encoding
      of the child identifier.  The above example would be named:

         '802-1Q 0x05000001'.  "
   DECODING
      "VLAN headers and tagged frame structure are defined in
      [IEEE802.1Q]."
   REFERENCE
      "The 802.1Q Protocol is defined in the Draft Standard for Virtual
      Bridged Local Area Networks [IEEE802.1Q]."
   ::= {
       ether2 0x8100       -- Ethernet or SNAP encoding of TPID
       -- snap 0x8100      ** excluded to reduce PD size & complexity
   }
```

5. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to

obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat."

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

6. Acknowledgements

This document was produced by the IETF RMONMIB Working Group.

The authors wish to thank the following people for their contributions to this document:

Anil Singhal
Frontier Software Development, Inc.

Jeanne Haney
Bay Networks

Dan Hansen
Network General Corp.

Special thanks are in order to the following people for writing RMON PI macro compilers, and improving the specification of the PI macro language:

David Perkins
DeskTalk Systems, Inc.

Skip Koppenhaver
Technically Elite, Inc.

7. References

[AF-LANE-0021.000] LAN Emulation Sub-working Group, B. Ellington, "LAN Emulation over ATM - Version 1.0", AF-LANE-0021.000, ATM Forum, IBM, January 1995.

[AF-NM-TEST-0080.000] Network Management Sub-working Group, Test Sub-working Group, A. Bierman, "Remote Monitoring MIB Extensions for ATM Networks", AF- NM-TEST-0080.000, ATM Forum, Cisco Systems, February 1997.

[IEEE802.1D-1998] LAN MAN Standards Committee of the IEEE Computer Society, "Information technology -- Telecommunications and information exchange between systems -- Local and metropolitan area networks -- Common specification -- Part 3: Media Access Control (MAC) Bridges", ISO/IEC Final DIS 15802-3 (IEEE P802.1D/D17) Institute of Electrical and Electronics Engineers, Inc., May 1998.

[IEEE802.1Q] LAN MAN Standards Committee of the IEEE Computer Society, "IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks", Draft Standard P802.1Q/D11, Institute of Electrical and Electronics Engineers, Inc., July 1998.

[RFC1155] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1212] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[RFC1215] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[RFC1483] Heinanen, J., "Multiprotocol Encapsulation over ATM Adaptation Layer 5", RFC 1483, July 1993.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[RFC1901] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC1902] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1902, January 1996.

[RFC1903] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1903, January 1996.

[RFC1904] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1904, January 1996.

[RFC1905] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)"", RFC 1906, January 1996.

[RFC2021] Waldbusser, S., "Remote Network Monitoring MIB (RMON-2)", RFC 2021, January 1997.

[RFC2074] Bierman, A. and R. Iddon, "Remote Network Monitoring MIB Protocol Identifiers", RFC 2074, January 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2233] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB Using SMIv2", RFC 2233, November 1997.

[RFC2271] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2271, January 1998.

[RFC2272] Case, J., Harrington D., Presuhn R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2272, January 1998.

[RFC2273] Levi, D., Meyer, P. and B. Stewart, "SNMPv3 Applications", RFC 2273, January 1998.

[RFC2274] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2274, January 1998.

[RFC2275] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2275, January 1998.

[RFC2570] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, April 1999.

[RFC2571] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2572] Case, J., Harrington D., Presuhn R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, April 1999.

[RFC2573] Levi, D., Meyer, P. and B. Stewart, "SNMPv3 Applications", RFC 2573, April 1999.

[RFC2574] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, April 1999.

[RFC2575] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC2896] Bierman, A., Bucci, C. and R. Iddon, "Remote Network Monitoring MIB Protocol Identifier Macros", RFC 2896, August 2000.

8. IANA Considerations

The protocols identified in this specification are almost entirely defined in external documents. In some rare cases, an arbitrary Protocol Identifier assignment must be made in order to support a particular protocol in the RMON-2 protocolDirTable. Protocol Identifier macros for such protocols will be defined under the 'ianaAssigned' base layer (see sections 3. and 4.2).

At this time, only one protocol is defined under the ianaAssigned base layer, called 'ipxOverRaw8023' (see section 4.2).

9. Security Considerations

This document discusses the syntax and semantics of textual descriptions of networking protocols, not the definition of any networking behavior. As such, no security considerations are raised by this memo.

10. Authors' Addresses

Andy Bierman
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA USA 95134

Phone: +1 408-527-3711
EMail: abierman@cisco.com

Chris Bucci
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA USA 95134

Phone: +1 408-527-5337
EMail: cbucci@cisco.com

Robin Iddon
c/o 3Com Inc.
Blackfriars House
40/50 Blackfrias Street
Edinburgh, EH1 1NE, UK

Phone: +44 131.558.3888
EMail: None

Appendix A: Changes since RFC 2074

The differences between RFC 2074 and this document are:

- RFC 2074 has been split into a reference document (this document) on the standards track and an informational document [RFC2896], in order to remove most protocol identifier macros out of the standards track document.
- Administrative updates; added an author, added copyrights, updated SNMP framework boilerplate;
- Updated overview section.
- Section 2.1 MUST, SHOULD text added per template
- Section 2.1 added some new terms
  - parent protocol
  - child protocol
  - protocol encapsulation tree
- Added section 2.3 about splitting into 2 documents:

  "Relationship to the RMON Protocol Identifier Macros Document"
- Added section 2.4 "Relationship to the ATM-RMON MIB"
- rewrote section 3.2 "Protocol Identifier Macro Format" But no semantic changes were made; The PI macro syntax is now specified in greater detail using BNF notation.
- Section 3.2.3.1 "Mapping of the 'countsFragments(0)' BIT"
  - this section was clarified to allow multiple protocolDirParameters octets in a given PI string to set the 'countsFragments' bit. The RFC version says just one octet can set this BIT. It is a useful feature to identify fragmentation at multiple layers, and most RMON-2 agents were already doing this, so the WG agreed to this clarification.
- Added section 4.3 "Encapsualtion Layers"
- This document ends after the base layer encapsulation definitions (through RFC 2074, section 5.2)
- Added Intellectual Property section
- Moved RFC 2074 section 5.3 "L3: Children of Base Protocol Identifiers" through the end of RFC 2074, to the PI Reference [RFC2896] document, in which many new protocol identifier macros were added for application protocols and non-IP protocol stacks.
- Acknowledgements section has been updated

11. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.