                 Distributed Management Expression MIB

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   This memo defines a portion of the Management Information Base (MIB)
   for use with network management protocols in the Internet community.
   In particular, it describes managed objects used for managing
   expressions of MIB objects. The results of these expressions become
   MIB objects usable like any other MIB object, such as for the test
   condition for declaring an event.

   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.

Table of Contents

1. The SNMP Management Framework

   The SNMP Management Framework presently consists of five major
   components:

   o   An overall architecture, described in RFC 2571 [RFC2571].

   o   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].

   o   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].

   o   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].

o   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 specifies a MIB module that is compliant to the SMIv2. A
   MIB conforming to the SMIv1 can be produced through the appropriate
   translations. The resulting translated MIB must be semantically
   equivalent, except where objects or events are omitted because no
   translation is possible (use of Counter64). Some machine readable
   information in SMIv2 will be converted into textual descriptions in
   SMIv1 during the translation process. However, this loss of machine
   readable information is not considered to change the semantics of the
   MIB.

2. Overview

   Users of MIBs often desire MIB objects that MIB designers have not
   provided. Furthermore, such needs vary from one management
   philosophy to another. Rather than fill more and more MIBs with
   standardized objects, the Expression MIB supports externally defined
   expressions of existing MIB objects.

   In the Expression MIB the results of an evaluated expression are MIB
   objects that may be used like any other MIB objects. These custom-
   defined objects are thus usable anywhere any other MIB object can be
   used. For example, they can be used by a management application
   directly or referenced from another MIB, such as the Event MIB
   [MIBEventMIB]. They can even be used by the Expression MIB itself,
   forming expressions of expressions.

   The Expression MIB is instrumentation for a relatively powerful,
   complex, high-level application, considerably different from simple
   instrumentation for a communication driver or a protocol. The MIB is
   appropriate in a relatively powerful, resource-rich managed system
   and not necessarily in a severely limited environment.

   Nevertheless, due to dependencies from the Event MIB [RFC2981] and
   the need to support as low-end a system as possible, the Expression
   MIB can be somewhat stripped down for lower-power, lower-resource
   implementations, as described in the Subsets section, below.

Implementation of the Expression MIB in a managed system led to the addition of objects that may not have been necessary in an application environment with complete knowledge of compiled MIB definitions. This is appropriate since implementation must be possible within typical managed systems with some constraints on system resources.

2.1. Usage

On managed systems that can afford the overhead, the Expression MIB is a way to create new, customized MIB objects for monitoring. Although these can save some network traffic and overhead on management systems, that is often not a good tradeoff for objects that are simply to be recorded or displayed.

An example of a use of the Expression MIB would be to provide custom objects for the Event MIB [RFC2981]. A complex expression can evaluate to a rate of flow or a boolean and thus be subject to testing as an event trigger, resulting in an SNMP notification. Without these capabilities such monitoring would be limited to the objects in predefined MIBs. The Expression MIB thus supports powerful tools for the network manager faced with the monitoring of large, complex systems that can support a significant level of self management.

2.2. Persistence

Although like most MIBs this one has no explicit controls for the persistence of the values set in configuring an expression, a robust, polite implementation would certainly not force its managing applications to reconfigure it whenever it resets.

Again, as with most MIBs, it is implementation specific how a system provides and manages such persistence. To speculate, one could imagine, for example, that persistence depended on the context in which the expression was configured, or perhaps system-specific characteristics of the expression's owner. Or perhaps everything in a MIB such as this one, which is clearly aimed at persistent configuration, is automatically part of a system's other persistent configuration.

2.3. Operation

Most of the operation of the MIB is described or implied in the object definitions but a few highlights bear mentioning here.

2.3.1. Sampling

   The MIB supports three types of object sampling for the MIB objects
   that make up the expression: absolute, delta, and changed.

   Absolute samples are simply the value of the MIB object at the time
   it is sampled.

   Absolute samples are not sufficient for expressions of counters, as
   counters have meaning only as a delta (difference) from one sample to
   the next. Thus objects may be sampled as deltas. Delta sampling
   requires the application to maintain state for the value at the last
   sample, and to do continuous sampling whether or not anyone is
   looking at the results. It thus creates constant overhead.

   Changed sampling is a simple fallout of delta sampling where rather
   than a difference the result is a boolean indicating whether or not
   the object changed value since the last sample.

2.3.2. Wildcards

   Wildcards allow the application of a single expression to multiple
   instances of the same MIB object. The definer of the expression
   indicates this choice and provides a partial object identifier, with
   some or all of the instance portion left off. The application then
   does the equivalent of GetNext to obtain the object values, thus
   discovering the instances.

   All wildcarded objects in an expression must have the same semantics
   for the missing portion of their object identifiers. Otherwise, any
   successful evaluation of the wildcarded expression would be the
   result of the accidental matching of the wildcarded portion of the
   object identifiers in the expression. Such an evaluation will likely
   produce results which are not meaningful.

   The expression can be evaluated only for those instances where all
   the objects in the expression are available with the same value for
   the wildcarded portion of the instance.

2.3.3. Evaluation

   There are two important aspects of evaluation that may not be
   obvious: what objects and when.

   What objects get used in the evaluation depends on the type of
   request and whether or not the expression contains wildcarded
   objects. If the request was a Get, that locks down the instances to

be used. If the request was a GetNext or GetBulk, the application
must work its way up to the next full set of objects for the
expression.

Evaluation of expressions happens at two possible times, depending on
the sampling method (delta or absolute) used to evaluate the
expression.

If there are no delta or change values in an expression, the
evaluation occurs on demand, i.e. when a requester attempts to read
the value of the expression. In this case all requesters get a
freshly calculated value.

For expressions with delta or change values, evaluation goes on
continuously, every sample period. In this case requesters get the
value as of the last sample period. For any given sample period of a
given expression, only those instances exist that provided a full set
of object values. It may be possible that a delta expression which
was evaluated successfully for one sample period may not be
successfully evaluated in the next sample period. This may, for
example, be due to missing instances for some or all of the objects
in the expression. In such cases, the value from the previous sample
period (with the successful evaluation) must not be carried forward
to the next sample period (with the failed evaluation).

2.3.4. Value Identification

Values resulting from expression evaluation are identified with a
combination of the object identifier (OID) for the data type from
expValueTable (such as expValueCounter32Val), the expression owner,
the expression name, and an OID fragment.

The OID fragment is not an entire OID beginning with iso.dod.org
(1.3.6). Rather it begins with 0.0. The remainder is either another
0 when there is no wildcarding or the instance that satisfied the
wildcard if there is wildcarding.

2.4. Subsets

To pare down the Expression MIBs complexity and use of resources an
implementor can leave out various parts.

2.4.1. No Wildcards

Leaving out wildcarding significantly reduces the complexity of
retrieving values to evaluate expressions and the processing required
to do so. Such an implementation would allow expressions made up of

individual MIB objects but would not be suitable for expressions
applied across large tables as each instance in the table would
require a separate expression definition.

Furthermore it would not be suitable for tables with arbitrary,
dynamic instances, as expressions definitions could not predict what
instance values to use.

An implementation without wildcards might be useful for a self-
managing system with small tables or few dynamic instances, or one
that can do calculations only for a few key objects.

2.4.2. No Deltas

Leaving out delta processing significantly reduces state that must be
kept and the burden of ongoing processing even when no one is looking
at the results. Unfortunately it also makes expressions on counters
unusable, as counters have meaning only as deltas.

An implementation without deltas might be useful for a severely
limited, self-managing system that has no need for expressions or
events on counters. Although conceivable, such systems would be
rare.

2.5. Structure

The MIB has the following sections:

    o   Resource -- management of the MIB's use of system resources.

    o   Definition -- definition of expressions.

    o   Value -- values of evaluated expressions.

2.5.1. Resource

The resource section has objects to manage resource usage by
wildcarded delta expressions, a potential major consumer of CPU and
memory.

2.5.2. Definition

The definition section contains the tables that define expressions.

The expression table, indexed by expression owner and expression
name, contains those parameters that apply to the entire expression,
such as the expression itself, the data type of the result, and the
sampling interval if it contains delta or change values.

The object table, indexed by expression owner, expression name and
object index within each expression, contains the parameters that
apply to the individual objects that go into the expression,
including the object identifier, sample type, discontinuity
indicator, and such.

2.5.3. Value

The value section contains the values of evaluated expressions.

The value table, indexed by expression owner, expression name and
instance fragment contains a "discriminated union" of evaluated
expression results. For a given expression only one of the columns
is instantiated, depending on the result data type for the
expression. The instance fragment is a constant or the final section
of the object identifier that filled in a wildcard.

2.6. Examples

The examples refer to tables and objects defined below in the MIB
itself. They may well make more sense after reading those
definitions.

2.6.1. Wildcarding

An expression may use wildcarded MIB objects that result in multiple
values for the expression. To specify a wildcarded MIB object a
management application leaves off part or all of the instance portion
of the object identifier, and sets expObjectWildcard to true(1) for
that object. For our example we'll use a counter of total blessings
from a table of people. Another table, indexed by town and person
has blessings just from that town.

So the index clauses are:

```
    personEntry OBJECT-TYPE
    ...
    INDEX { personIndex }
```

And:

```
    townPersonEntry OBJECT-TYPE
    ...
    INDEX { townIndex, personIndex }
```

In our friendly application we may have entered our expression as:

    100 * townPersonBlessings.976.* / personBlessings.*

What goes in expExpression is:

    100*$1/$2

For example purposes we'll use some slightly far-fetched OIDs. The
People MIB is 1.3.6.1.99.7 and the Town MIB is 1.3.6.1.99.11, so for
our two counters the OIDs are:

    personBlessings   1.3.6.1.99.7.1.3.1.4
    townPersonBlessings   1.3.6.1.99.11.1.2.1.9

The rule for wildcards is that all the wildcarded parts have to match
exactly. In this case that means we have to hardwire the town and
only the personIndex can be wildcarded. So our values for
expObjectID are:

    1.3.6.1.99.7.1.3.1.4
    1.3.6.1.99.11.1.2.1.9.976

We're hardwired to townIndex 976 and personIndex is allowed to vary.

The value of expExpressionPrefix can be either of those two counter
OIDs (including the instance fragment in the second case), since
either of them takes you to a MIB definition where you can look at
the INDEX clause and figure out what's been left off. What's been
left off doesn't have to work out to be the same object, but it does
have to work out to be the same values (semantics) for the result to
make sense. Note that the managed system can not typically check
such semantics and if given nonsense will return nonsense.

If we have people numbered 6, 19, and 42 in town number 976, the
successive values of expValueInstance will be:

    0.0.6
    0.0.19
    0.0.42

So there will be three values in expValueTable, with those OIDs as
the expValueInstance part of their indexing.

2.6.2. Calculation and Conditional

   The following formula for line utilization of a half-duplex link is
   adapted from [PracPersp].

    utilization = (ifInOctets + ifOutOctets) * 800 / seconds / ifSpeed

   The expression results in the percentage line utilization per second.
   The total octets are multiplied by 8 to get bits and 100 to scale up
   the percentage as an integer.

   The following Expression MIB object values implement this as an
   expression for all ifIndexes that directly represent actual hardware.
   Since the octet counters are Counter32 values, they must be delta
   sampled to be meaningful. The sample period is 6 seconds but for
   accuracy and independence is calculated as a delta of sysUpTime.

   The expObjectTable entry for ifInOctets has an expObjectConditional
   that checks for being a hardware interface. Only one object in the
   expression needs that check associated, since it applies to the whole
   expression. Since ifConnectorPresent is a TruthValue with values of
   1 or 2 rather than 0 and non-zero, it must also be in an expression
   rather than used directly for the conditional.

   The interface-specific discontinuity indicator is supplied only for
   ifInOctets since invalidating that sample will invalidate an attempt
   at evaluation, effectively invalidating ifOutOctets as well
   (correctly, because it has the same indicator).

   For notational clarity, in the rest of this document, a string in
   quotes as part of the object instance indicates the value that would
   actually be one subidentifier per byte. The objects all belong to
   owner "me".

   Also for clarity OIDs are expressed as the object descriptor and
   instance. In fact they must be supplied numerically, with all
   subidentifiers in place before the part for the particular object and
   instance.

   What the user would set in expExpressionTable:

   expExpression.2."me".4."hard"   = "$1==1"
   expExpressionValueType.2."me".4."hard" = unsigned32
   expExpressionRowStatus.2."me"4."hard"   = 'active'

```
   expExpression.2."me".4."util"        = "($1+$2)*800/$4/$3"
   expExpressionValueType.2."me".4."util" = integer32
   expExpressionDeltaInterval.2."me".4."util"  = 6
   expExpressionRowStatus.2."me"4."util"   = 'active'
```

What the user would set in expObjectTable:

```
   expObjectID.2."me".4."hard".1       = ifConnectorPresent
   expObjectWildcard.2."me".4."hard".1     = 'true'
   expObjectSampleType.2."me".4."hard".1   = 'absoluteValue'
   expObjectRowStatus.2."me".4."hard".1    = 'active'

   expObjectID.2."me".4."util".1       = ifInOctets
   expObjectWildcard.2."me".4."util".1     = 'true'
   expObjectSampleType.2."me".4."util".1   = 'deltaValue'
   expObjectConditional.2."me".4."util".1  =
   expValueUnsigned32Val.4."hard".0.0
   expObjectConditionalWildcard.2."me".4."util".1    = 'true'
   expObjectDiscontinuityID.2."me".4."util".1   =
   ifCounterDiscontinuityTime
   expObjectDiscontinuityIDWildcard.2."me".4."util".1     = 'true'
   expObjectRowStatus.2."me".4."util".1    = 'active'

   expObjectID.2."me".4."util".2       = ifOutOctets
   expObjectWildcard.2."me".4."util".2     = 'true'
   expObjectSampleType.2."me".4."util".2   = 'deltaValue'
   expObjectRowStatus.2."me".4."util".2    = 'active'

   expObjectID.2."me".4."util".3       = ifSpeed
   expObjectWildcard.2."me".4."util".3     = 'true'
   expObjectSampleType.2."me".4."util".3   = 'absoluteValue'
   expObjectRowStatus.2."me".4."util".3    = 'active'

   expObjectID.2."me".4."util".4       = sysUpTime.0
   expObjectWildcard.2."me".4."util".4     = 'false'
   expObjectSampleType.2."me".4."util".4   = 'deltaValue'
   expObjectRowStatus.2."me".4."util".4    = 'active'
```

These settings will result in populating one column of expValueTable:

```
   expValueInteger32Val.2."me".4."util".0.0.?
```

The subidentifier represented by "?" above represents one
subidentifier that takes on a value of ifIndex and identifies a row
for each ifIndex value where ifConnectorPresent is 'true' and the
interface was present for two samples to provide a delta.

   This value could in turn be used as an event threshold [RFC2981] to
   watch for overutilization of all hardware network connections.

3. Definitions

DISMAN-EXPRESSION-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, Gauge32, Unsigned32,
    Counter32, Counter64, IpAddress,
    TimeTicks, mib-2, zeroDotZero   FROM SNMPv2-SMI
    RowStatus, TruthValue, TimeStamp   FROM SNMPv2-TC
    sysUpTime   FROM SNMPv2-MIB
    SnmpAdminString   FROM SNMP-FRAMEWORK-MIB
    MODULE-COMPLIANCE, OBJECT-GROUP   FROM SNMPv2-CONF;

dismanExpressionMIB MODULE-IDENTITY
    LAST-UPDATED "200010160000Z" -- 16 October 2000
    ORGANIZATION "IETF Distributed Management Working Group"
    CONTACT-INFO "Ramanathan Kavasseri
                  Cisco Systems, Inc.
                  170 West Tasman Drive,
                  San Jose CA 95134-1706.
                  Phone: +1 408 527 2446
                  Email: ramk@cisco.com"
    DESCRIPTION
     "The MIB module for defining expressions of MIB objects for
     management purposes."
-- Revision History

        REVISION   "200010160000Z" -- 16 October 2000
        DESCRIPTION "This is the initial version of this MIB.
                     Published as RFC 2982"
     ::= { mib-2 90 }

dismanExpressionMIBObjects OBJECT IDENTIFIER ::=
                                          { dismanExpressionMIB 1 }

expResource   OBJECT IDENTIFIER ::= { dismanExpressionMIBObjects 1 }
expDefine OBJECT IDENTIFIER ::= { dismanExpressionMIBObjects 2 }
expValue OBJECT IDENTIFIER ::= { dismanExpressionMIBObjects 3 }

--
-- Resource Control
--

expResourceDeltaMinimum OBJECT-TYPE
    SYNTAX   Integer32 (-1 | 1..600)
    UNITS   "seconds"
    MAX-ACCESS   read-write
    STATUS   current
    DESCRIPTION
     "The minimum expExpressionDeltaInterval this system will
     accept. A system may use the larger values of this minimum to
     lessen the impact of constantly computing deltas. For larger
     delta sampling intervals the system samples less often and
     suffers less overhead. This object provides a way to enforce
     such lower overhead for all expressions created after it is
     set.

     The value -1 indicates that expResourceDeltaMinimum is
     irrelevant as the system will not accept 'deltaValue' as a
     value for expObjectSampleType.

     Unless explicitly resource limited, a system's value for
     this object should be 1, allowing as small as a 1 second
     interval for ongoing delta sampling.

     Changing this value will not invalidate an existing setting
     of expObjectSampleType."
    ::= { expResource 1 }

expResourceDeltaWildcardInstanceMaximum OBJECT-TYPE
    SYNTAX   Unsigned32
    UNITS   "instances"
    MAX-ACCESS   read-write
    STATUS   current
    DESCRIPTION
     "For every instance of a deltaValue object, one dynamic instance
     entry is needed for holding the instance value from the previous
     sample, i.e. to maintain state.

     This object limits maximum number of dynamic instance entries
     this system will support for wildcarded delta objects in
     expressions. For a given delta expression, the number of
     dynamic instances is the number of values that meet all criteria
     to exist times the number of delta values in the expression.

     A value of 0 indicates no preset limit, that is, the limit
     is dynamic based on system operation and resources.

     Unless explicitly resource limited, a system's value for
     this object should be 0.

Changing this value will not eliminate or inhibit existing delta
wildcard instance objects but will prevent the creation of more
such objects.

An attempt to allocate beyond the limit results in expErrorCode
being tooManyWildcardValues for that evaluation attempt."
::= { expResource 2 }

expResourceDeltaWildcardInstances OBJECT-TYPE
    SYNTAX   Gauge32
    UNITS   "instances"
    MAX-ACCESS   read-only
    STATUS   current
    DESCRIPTION
     "The number of currently active instance entries as
     defined for expResourceDeltaWildcardInstanceMaximum."
    ::= { expResource 3 }

expResourceDeltaWildcardInstancesHigh OBJECT-TYPE
    SYNTAX   Gauge32
    UNITS   "instances"
    MAX-ACCESS   read-only
    STATUS   current
    DESCRIPTION
     "The highest value of expResourceDeltaWildcardInstances
     that has occurred since initialization of the managed
     system."
    ::= { expResource 4 }

expResourceDeltaWildcardInstanceResourceLacks OBJECT-TYPE
    SYNTAX   Counter32
    UNITS   "instances"
    MAX-ACCESS   read-only
    STATUS   current
    DESCRIPTION
     "The number of times this system could not evaluate an
     expression because that would have created a value instance in
     excess of expResourceDeltaWildcardInstanceMaximum."
    ::= { expResource 5 }

--

-- Definition
--
-- Expression Definition Table
--

expExpressionTable OBJECT-TYPE

```
        SYNTAX       SEQUENCE OF ExpExpressionEntry
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
         "A table of expression definitions."
        ::= { expDefine 1 }

expExpressionEntry OBJECT-TYPE
        SYNTAX       ExpExpressionEntry
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
         "Information about a single expression.  New expressions
         can be created using expExpressionRowStatus.

         To create an expression first create the named entry in this
         table.  Then use expExpressionName to populate expObjectTable.
         For expression evaluation to succeed all related entries in
         expExpressionTable and expObjectTable must be 'active'.  If
         these conditions are not met the corresponding values in
         expValue simply are not instantiated.

         Deleting an entry deletes all related entries in expObjectTable
         and expErrorTable.

         Because of the relationships among the multiple tables for an
         expression (expExpressionTable, expObjectTable, and
         expValueTable) and the SNMP rules for independence in setting
         object values, it is necessary to do final error checking when
         an expression is evaluated, that is, when one of its instances
         in expValueTable is read or a delta interval expires.  Earlier
         checking need not be done and an implementation may not impose
         any ordering on the creation of objects related to an
         expression.

         To maintain security of MIB information, when creating a new row in
         this table, the managed system must record the security credentials
         of the requester.  These security credentials are the parameters
         necessary as inputs to isAccessAllowed from the Architecture for

         Describing SNMP Management Frameworks.  When obtaining the objects
         that make up the expression, the system must (conceptually) use
         isAccessAllowed to ensure that it does not violate security.

         The evaluation of the expression takes place under the
         security credentials of the creator of its expExpressionEntry.

         Values of read-write objects in this table may be changed
```

```
          at any time."
    INDEX        { expExpressionOwner, expExpressionName }
    ::= { expExpressionTable 1 }


ExpExpressionEntry ::= SEQUENCE {
    expExpressionOwner            SnmpAdminString,
    expExpressionName             SnmpAdminString,
    expExpression                 OCTET STRING,
    expExpressionValueType        INTEGER,
    expExpressionComment          SnmpAdminString,
    expExpressionDeltaInterval    Integer32,
    expExpressionPrefix           OBJECT IDENTIFIER,
    expExpressionErrors           Counter32,
    expExpressionEntryStatus      RowStatus
}

expExpressionOwner OBJECT-TYPE
    SYNTAX        SnmpAdminString (SIZE(0..32))
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
      "The owner of this entry. The exact semantics of this
       string are subject to the security policy defined by the
       security administrator."
      ::= { expExpressionEntry 1 }

expExpressionName OBJECT-TYPE
    SYNTAX        SnmpAdminString (SIZE (1..32))
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
      "The name of the expression.  This is locally unique, within
       the scope of an expExpressionOwner."
      ::= { expExpressionEntry 2 }

expExpression OBJECT-TYPE
    SYNTAX        OCTET STRING (SIZE (1..1024))
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
      "The expression to be evaluated.  This object is the same
       as a DisplayString (RFC 1903) except for its maximum length.

       Except for the variable names the expression is in ANSI C
       syntax.  Only the subset of ANSI C operators and functions
       listed here is allowed.

       Variables are expressed as a dollar sign ('$') and an
```

integer that corresponds to an expObjectIndex. An
example of a valid expression is:

        ($1-$5)*100

Expressions must not be recursive, that is although an expression
may use the results of another expression, it must not contain
any variable that is directly or indirectly a result of its own
evaluation. The managed system must check for recursive
expressions.

The only allowed operators are:

        ( )
        - (unary)
        + - * / %
        & | ^ << >> ~
        ! && || == != > >= < <=

Note the parentheses are included for parenthesizing the
expression, not for casting data types.

The only constant types defined are:

        int (32-bit signed)
        long (64-bit signed)
        unsigned int
        unsigned long
        hexadecimal
        character
        string
        oid

The default type for a positive integer is int unless it is too
large in which case it is long.

All but oid are as defined for ANSI C. Note that a
hexadecimal constant may end up as a scalar or an array of
8-bit integers. A string constant is enclosed in double
quotes and may contain back-slashed individual characters
as in ANSI C.

An oid constant comprises 32-bit, unsigned integers and at
least one period, for example:

        0.
        .0
        1.3.6.1

No additional leading or trailing subidentifiers are automatically
added to an OID constant. The constant is taken as expressed.

Integer-typed objects are treated as 32- or 64-bit, signed
or unsigned integers, as appropriate. The results of
mixing them are as for ANSI C, including the type of the
result. Note that a 32-bit value is thus promoted to 64 bits
only in an operation with a 64-bit value. There is no
provision for larger values to handle overflow.

Relative to SNMP data types, a resulting value becomes
unsigned when calculating it uses any unsigned value,
including a counter. To force the final value to be of
data type counter the expression must explicitly use the
counter32() or counter64() function (defined below).

OCTET STRINGS and OBJECT IDENTIFIERs are treated as
one-dimensional arrays of unsigned 8-bit integers and
unsigned 32-bit integers, respectively.

IpAddresses are treated as 32-bit, unsigned integers in
network byte order, that is, the hex version of 255.0.0.0 is
0xff000000.

Conditional expressions result in a 32-bit, unsigned integer
of value 0 for false or 1 for true. When an arbitrary value
is used as a boolean 0 is false and non-zero is true.

Rules for the resulting data type from an operation, based on
the operator:

For << and >> the result is the same as the left hand operand.

For &&, ||, ==, !=, <, <=, >, and >= the result is always
Unsigned32.

For unary - the result is always Integer32.

For +, -, *, /, %, &, |, and ^ the result is promoted according
to the following rules, in order from most to least preferred:

    If left hand and right hand operands are the same type,
    use that.

    If either side is Counter64, use that.

    If either side is IpAddress, use that.

If either side is TimeTicks, use that.

If either side is Counter32, use that.

Otherwise use Unsigned32.

The following rules say what operators apply with what data types. Any combination not explicitly defined does not work.

For all operators any of the following can be the left hand or right hand operand: Integer32, Counter32, Unsigned32, Counter64.

The operators +, -, *, /, %, <, <=, >, and >= work with TimeTicks.

The operators &, |, and ^ work with IpAddress.

The operators << and >> work with IpAddress but only as the left hand operand.

The + operator performs a concatenation of two OCTET STRINGs or two OBJECT IDENTIFIERs.

The operators &, | perform bitwise operations on OCTET STRINGs. If the OCTET STRING happens to be a DisplayString the results may be meaningless, but the agent system does not check this as some such systems do not have this information.

The operators << and >> perform bitwise operations on OCTET STRINGs appearing as the left hand operand.

The only functions defined are:

        counter32
        counter64
        arraySection
        stringBegins
        stringEnds
        stringContains
        oidBegins
        oidEnds
        oidContains
        average
        maximum
        minimum
        sum
        exists

   The following function definitions indicate their parameters by
   naming the data type of the parameter in the parameter's position
   in the parameter list. The parameter must be of the type indicated
   and generally may be a constant, a MIB object, a function, or an
   expression.

   counter32(integer) - wrapped around an integer value counter32
   forces Counter32 as a data type.

   counter64(integer) - similar to counter32 except that the
   resulting data type is 'counter64'.

   arraySection(array, integer, integer) - selects a piece of an
   array (i.e. part of an OCTET STRING or OBJECT IDENTIFIER). The
   integer arguments are in the range 0 to 4,294,967,295. The
   first is an initial array index (one-dimensioned) and the second
   is an ending array index. A value of 0 indicates first or last
   element, respectively. If the first element is larger than the
   array length the result is 0 length. If the second integer is
   less than or equal to the first, the result is 0 length. If the
   second is larger than the array length it indicates last
   element.

   stringBegins/Ends/Contains(octetString, octetString) - looks for
   the second string (which can be a string constant) in the first
   and returns the one-dimensioned arrayindex where the match began.
   A return value of 0 indicates no match (i.e. boolean false).

   oidBegins/Ends/Contains(oid, oid) - looks for the second OID
   (which can be an OID constant) in the first and returns the
   the one-dimensioned index where the match began. A return value
   of 0 indicates no match (i.e. boolean false).

   average/maximum/minimum(integer) - calculates the average,
   minimum, or maximum value of the integer valued object over
   multiple sample times. If the object disappears for any
   sample period, the accumulation and the resulting value object
   cease to exist until the object reappears at which point the
   calculation starts over.

   sum(integerObject*) - sums all available values of the
   wildcarded integer object, resulting in an integer scalar. Must
   be used with caution as it wraps on overflow with no
   notification.

   exists(anyTypeObject) - verifies the object instance exists. A
   return value of 0 indicates NoSuchInstance (i.e. boolean
   false)."

```
     ::= { expExpressionEntry 3 }

expExpressionValueType OBJECT-TYPE
     SYNTAX       INTEGER { counter32(1), unsigned32(2), timeTicks(3),
                   integer32(4), ipAddress(5), octetString(6),
                   objectId(7), counter64(8) }
     MAX-ACCESS  read-create
     STATUS      current
     DESCRIPTION
      "The type of the expression value.  One and only one of the
      value objects in expValueTable will be instantiated to match
      this type.

      If the result of the expression can not be made into this type,
      an invalidOperandType error will occur."
     DEFVAL      { counter32 }
     ::= { expExpressionEntry 4 }

expExpressionComment OBJECT-TYPE
     SYNTAX       SnmpAdminString
     MAX-ACCESS  read-create
     STATUS      current
     DESCRIPTION
      "A comment to explain the use or meaning of the expression."
     DEFVAL      { ''H }
     ::= { expExpressionEntry 5 }

expExpressionDeltaInterval OBJECT-TYPE
     SYNTAX       Integer32 (0..86400)
     UNITS       "seconds"
     MAX-ACCESS  read-create
     STATUS      current
     DESCRIPTION
      "Sampling interval for objects in this expression with
      expObjectSampleType 'deltaValue'.

      This object has no effect if the the expression has no
      deltaValue objects.

      A value of 0 indicates no automated sampling.  In this case
      the delta is the difference from the last time the expression
      was evaluated.  Note that this is subject to unpredictable
      delta times in the face of retries or multiple managers.

      A value greater than zero is the number of seconds between
      automated samples.

      Until the delta interval has expired once the delta for the
```

          object is effectively not instantiated and evaluating
          the expression has results as if the object itself were not
          instantiated.

          Note that delta values potentially consume large amounts of
          system CPU and memory. Delta state and processing must
          continue constantly even if the expression is not being used.
          That is, the expression is being evaluated every delta interval,
          even if no application is reading those values. For wildcarded
          objects this can be substantial overhead.

          Note that delta intervals, external expression value sampling
          intervals and delta intervals for expressions within other
          expressions can have unusual interactions as they are impossible
          to synchronize accurately. In general one interval embedded
          below another must be enough shorter that the higher sample
          sees relatively smooth, predictable behavior. So, for example,
          to avoid the higher level getting the same sample twice, the
          lower level should sample at least twice as fast as the higher
          level does."
        DEFVAL   { 0 }
        ::= { expExpressionEntry 6 }

    expExpressionPrefix OBJECT-TYPE
        SYNTAX   OBJECT IDENTIFIER
        MAX-ACCESS   read-only
        STATUS   current
        DESCRIPTION
          "An object prefix to assist an application in determining
          the instance indexing to use in expValueTable, relieving the
          application of the need to scan the expObjectTable to
          determine such a prefix.

          See expObjectTable for information on wildcarded objects.

          If the expValueInstance portion of the value OID may
          be treated as a scalar (that is, normally, 0) the value of
          expExpressionPrefix is zero length, that is, no OID at all.
          Note that zero length implies a null OID, not the OID 0.0.

          Otherwise, the value of expExpressionPrefix is the expObjectID
          value of any one of the wildcarded objects for the expression.
          This is sufficient, as the remainder, that is, the instance
          fragment relevant to instancing the values, must be the same for
          all wildcarded objects in the expression."
          ::= { expExpressionEntry 7 }

    expExpressionErrors OBJECT-TYPE

```
      SYNTAX        Counter32
      MAX-ACCESS    read-only
      STATUS        current
      DESCRIPTION
       "The number of errors encountered while evaluating this
       expression.

       Note that an object in the expression not being accessible,
       is not considered an error. An example of an inaccessible
       object is when the object is excluded from the view of the
       user whose security credentials are used in the expression
       evaluation. In such cases, it is a legitimate condition
       that causes the corresponding expression value not to be
       instantiated."
      ::= { expExpressionEntry 8 }

expExpressionEntryStatus OBJECT-TYPE
      SYNTAX        RowStatus
      MAX-ACCESS    read-create
      STATUS        current
      DESCRIPTION
       "The control that allows creation and deletion of entries."
      ::= { expExpressionEntry 9 }

--
-- Expression Error Table
--

expErrorTable OBJECT-TYPE
      SYNTAX        SEQUENCE OF ExpErrorEntry
      MAX-ACCESS    not-accessible
      STATUS        current
      DESCRIPTION
       "A table of expression errors."
      ::= { expDefine 2 }

expErrorEntry OBJECT-TYPE
      SYNTAX        ExpErrorEntry
      MAX-ACCESS    not-accessible
      STATUS        current
      DESCRIPTION
       "Information about errors in processing an expression.

       Entries appear in this table only when there is a matching
       expExpressionEntry and then only when there has been an
       error for that expression as reflected by the error codes
       defined for expErrorCode."
      INDEX         { expExpressionOwner, expExpressionName }
```

```
    ::= { expErrorTable 1 }

ExpErrorEntry ::= SEQUENCE {
    expErrorTime      TimeStamp,
    expErrorIndex     Integer32,
    expErrorCode      INTEGER,
    expErrorInstance  OBJECT IDENTIFIER
}

expErrorTime OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value of sysUpTime the last time an error caused a
      failure to evaluate this expression."
    ::= { expErrorEntry 1 }

expErrorIndex OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The one-dimensioned character array index into
      expExpression for where the error occurred.  The value
      zero indicates irrelevance."
    ::= { expErrorEntry 2 }

expErrorCode OBJECT-TYPE
    SYNTAX      INTEGER {
          invalidSyntax(1),
          undefinedObjectIndex(2),
          unrecognizedOperator(3),
          unrecognizedFunction(4),
          invalidOperandType(5),
          unmatchedParenthesis(6),
          tooManyWildcardValues(7),
          recursion(8),
          deltaTooShort(9),
          resourceUnavailable(10),
          divideByZero(11)
          }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The error that occurred.  In the following explanations the
      expected timing of the error is in parentheses.  'S' means
      the error occurs on a Set request.  'E' means the error
```

occurs on the attempt to evaluate the expression either due to
Get from expValueTable or in ongoing delta processing.

```
invalidSyntax         the value sent for expExpression is not
                      valid Expression MIB expression syntax
                      (S)
undefinedObjectIndex    an object reference ($n) in
                      expExpression does not have a matching
                      instance in expObjectTable (E)
unrecognizedOperator    the value sent for expExpression held an
                      unrecognized operator (S)
unrecognizedFunction    the value sent for expExpression held an
                      unrecognized function name (S)
invalidOperandType  an operand in expExpression is not the
                      right type for the associated operator
                      or result (SE)
unmatchedParenthesis    the value sent for expExpression is not
                      correctly parenthesized (S)
tooManyWildcardValues   evaluating the expression exceeded the
                      limit set by
                      expResourceDeltaWildcardInstanceMaximum
                      (E)
recursion        through some chain of embedded
                      expressions the expression invokes itself
                      (E)
deltaTooShort        the delta for the next evaluation passed
                      before the system could evaluate the
                      present sample (E)
resourceUnavailable some resource, typically dynamic memory,
                      was unavailable (SE)
divideByZero         an attempt to divide by zero occurred
                      (E)
```

For the errors that occur when the attempt is made to set
expExpression Set request fails with the SNMP error code
'wrongValue'. Such failures refer to the most recent failure to
Set expExpression, not to the present value of expExpression
which must be either unset or syntactically correct.

Errors that occur during evaluation for a Get* operation return
the SNMP error code 'genErr' except for 'tooManyWildcardValues'
and 'resourceUnavailable' which return the SNMP error code
'resourceUnavailable'."
::= { expErrorEntry 3 }

expErrorInstance OBJECT-TYPE
    SYNTAX   OBJECT IDENTIFIER
    MAX-ACCESS   read-only

        STATUS   current
        DESCRIPTION
         "The expValueInstance being evaluated when the error
          occurred. A zero-length indicates irrelevance."
        ::= { expErrorEntry 4 }

--
-- Object Table
--

expObjectTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF ExpObjectEntry
        MAX-ACCESS not-accessible
        STATUS   current
        DESCRIPTION
         "A table of object definitions for each expExpression.

          Wildcarding instance IDs:

          It is legal to omit all or part of the instance portion for
          some or all of the objects in an expression. (See the
          DESCRIPTION of expObjectID for details. However, note that
          if more than one object in the same expression is wildcarded
          in this way, they all must be objects where that portion of
          the instance is the same. In other words, all objects may be
          in the same SEQUENCE or in different SEQUENCEs but with the
          same semantic index value (e.g., a value of ifIndex)
          for the wildcarded portion."
        ::= { expDefine 3 }

expObjectEntry OBJECT-TYPE
        SYNTAX   ExpObjectEntry
        MAX-ACCESS not-accessible
        STATUS   current
        DESCRIPTION
         "Information about an object. An application uses
          expObjectEntryStatus to create entries in this table while
          in the process of defining an expression.

          Values of read-create objects in this table may be
          changed at any time."
        INDEX   { expExpressionOwner, expExpressionName, expObjectIndex }
        ::= { expObjectTable 1 }

ExpObjectEntry ::= SEQUENCE {
        expObjectIndex   Unsigned32,
        expObjectID   OBJECT IDENTIFIER,
        expObjectIDWildcard   TruthValue,

```
    expObjectSampleType                 INTEGER,
    expObjectDeltaDiscontinuityID       OBJECT IDENTIFIER,
    expObjectDiscontinuityIDWildcard    TruthValue,
    expObjectDiscontinuityIDType        INTEGER,
    expObjectConditional                OBJECT IDENTIFIER,
    expObjectConditionalWildcard        TruthValue,
    expObjectEntryStatus                RowStatus
}
```

expObjectIndex OBJECT-TYPE
    SYNTAX   Unsigned32 (1..4294967295)
    MAX-ACCESS   not-accessible
    STATUS   current
    DESCRIPTION
     "Within an expression, a unique, numeric identification for an
     object. Prefixed with a dollar sign ('$') this is used to
     reference the object in the corresponding expExpression."
    ::= { expObjectEntry 1 }

expObjectID OBJECT-TYPE
    SYNTAX   OBJECT IDENTIFIER
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
     "The OBJECT IDENTIFIER (OID) of this object. The OID may be
     fully qualified, meaning it includes a complete instance
     identifier part (e.g., ifInOctets.1 or sysUpTime.0), or it
     may not be fully qualified, meaning it may lack all or part
     of the instance identifier. If the expObjectID is not fully
     qualified, then expObjectWildcard must be set to true(1).
     The value of the expression will be multiple
     values, as if done for a GetNext sweep of the object.

     An object here may itself be the result of an expression but
     recursion is not allowed.

     NOTE: The simplest implementations of this MIB may not allow
     wildcards."
    ::= { expObjectEntry 2 }

expObjectIDWildcard OBJECT-TYPE
    SYNTAX   TruthValue
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
        "A true value indicates the expObjecID of this row is a wildcard
        object. False indicates that expObjectID is fully instanced.
        If all expObjectWildcard values for a given expression are FALSE,

          expExpressionPrefix will reflect a scalar object (i.e. will
          be 0.0).

          NOTE: The simplest implementations of this MIB may not allow
          wildcards."
      DEFVAL   { false }
      ::= { expObjectEntry 3 }

expObjectSampleType OBJECT-TYPE
      SYNTAX   INTEGER { absoluteValue(1), deltaValue(2),
                            changedValue(3) }
      MAX-ACCESS   read-create
      STATUS   current
      DESCRIPTION
       "The method of sampling the selected variable.

       An 'absoluteValue' is simply the present value of the object.

       A 'deltaValue' is the present value minus the previous value,
       which was sampled expExpressionDeltaInterval seconds ago.
       This is intended primarily for use with SNMP counters, which are
       meaningless as an 'absoluteValue', but may be used with any
       integer-based value.

       A 'changedValue' is a boolean for whether the present value is
       different from the previous value. It is applicable to any data
       type and results in an Unsigned32 with value 1 if the object's
       value is changed and 0 if not. In all other respects it is as a
       'deltaValue' and all statements and operation regarding delta
       values apply to changed values.

       When an expression contains both delta and absolute values
       the absolute values are obtained at the end of the delta
       period."
      DEFVAL   { absoluteValue }
      ::= { expObjectEntry 4 }

sysUpTimeInstance OBJECT IDENTIFIER ::= { sysUpTime 0 }

expObjectDeltaDiscontinuityID OBJECT-TYPE
      SYNTAX   OBJECT IDENTIFIER
      MAX-ACCESS   read-create
      STATUS   current
      DESCRIPTION
       "The OBJECT IDENTIFIER (OID) of a TimeTicks, TimeStamp, or
       DateAndTime object that indicates a discontinuity in the value
       at expObjectID.

This object is instantiated only if expObjectSampleType is
'deltaValue' or 'changedValue'.

The OID may be for a leaf object (e.g. sysUpTime.0) or may
be wildcarded to match expObjectID.

This object supports normal checking for a discontinuity in a
counter. Note that if this object does not point to sysUpTime
discontinuity checking must still check sysUpTime for an overall
discontinuity.

If the object identified is not accessible no discontinuity
check will be made."
```
    DEFVAL      { sysUpTimeInstance }
    ::= { expObjectEntry 5 }

expObjectDiscontinuityIDWildcard OBJECT-TYPE
    SYNTAX       TruthValue
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
```
"A true value indicates the expObjectDeltaDiscontinuityID of
this row is a wildcard object. False indicates that
expObjectDeltaDiscontinuityID is fully instanced.

This object is instantiated only if expObjectSampleType is
'deltaValue' or 'changedValue'.

NOTE: The simplest implementations of this MIB may not allow
wildcards."
```
    DEFVAL      { false }
    ::= { expObjectEntry 6 }

expObjectDiscontinuityIDType OBJECT-TYPE
    SYNTAX       INTEGER { timeTicks(1), timeStamp(2), dateAndTime(3) }
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
```
"The value 'timeTicks' indicates the expObjectDeltaDiscontinuityID
of this row is of syntax TimeTicks. The value 'timeStamp' indicates
syntax TimeStamp. The value 'dateAndTime indicates syntax
DateAndTime.

This object is instantiated only if expObjectSampleType is
'deltaValue' or 'changedValue'."
```
    DEFVAL      { timeTicks }
    ::= { expObjectEntry 7 }
```

expObjectConditional OBJECT-TYPE
    SYNTAX   OBJECT IDENTIFIER
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
     "The OBJECT IDENTIFIER (OID) of an object that overrides
     whether the instance of expObjectID is to be considered
     usable. If the value of the object at expObjectConditional
     is 0 or not instantiated, the object at expObjectID is
     treated as if it is not instantiated. In other words,
     expObjectConditional is a filter that controls whether or
     not to use the value at expObjectID.

     The OID may be for a leaf object (e.g. sysObjectID.0) or may be
     wildcarded to match expObjectID. If expObject is wildcarded and
     expObjectID in the same row is not, the wild portion of
     expObjectConditional must match the wildcarding of the rest of
     the expression. If no object in the expression is wildcarded
     but expObjectConditional is, use the lexically first instance
     (if any) of expObjectConditional.

     If the value of expObjectConditional is 0.0 operation is
     as if the value pointed to by expObjectConditional is a
     non-zero (true) value.

     Note that expObjectConditional can not trivially use an object
     of syntax TruthValue, since the underlying value is not 0 or 1."
    DEFVAL   { zeroDotZero }
    ::= { expObjectEntry 8 }

 expObjectConditionalWildcard OBJECT-TYPE
    SYNTAX   TruthValue
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION

    "A true value indicates the expObjectConditional of this row is
    a wildcard object. False indicates that expObjectConditional is
    fully instanced.

    NOTE: The simplest implementations of this MIB may not allow
    wildcards."
    DEFVAL   { false }
     ::= { expObjectEntry 9 }

expObjectEntryStatus OBJECT-TYPE
    SYNTAX   RowStatus
    MAX-ACCESS   read-create

        STATUS   current
        DESCRIPTION
         "The control that allows creation/deletion of entries.

         Objects in this table may be changed while
         expObjectEntryStatus is in any state."
        ::= { expObjectEntry 10 }

--
-- Expression Value Table
--

expValueTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF ExpValueEntry
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION
         "A table of values from evaluated expressions."
        ::= { expValue 1 }

expValueEntry OBJECT-TYPE
        SYNTAX   ExpValueEntry
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION
         "A single value from an evaluated expression. For a given
         instance, only one 'Val' object in the conceptual row will be
         instantiated, that is, the one with the appropriate type for
         the value. For values that contain no objects of
         expObjectSampleType 'deltaValue' or 'changedValue', reading a
         value from the table causes the evaluation of the expression
         for that value. For those that contain a 'deltaValue' or
         'changedValue' the value read is as of the last sampling
         interval.

         If in the attempt to evaluate the expression one or more
         of the necessary objects is not available, the corresponding
         entry in this table is effectively not instantiated.

         To maintain security of MIB information, when creating a new
         row in this table, the managed system must record the security
         credentials of the requester. These security credentials are
         the parameters necessary as inputs to isAccessAllowed from
         [RFC2571]. When obtaining the objects that make up the
         expression, the system must (conceptually) use isAccessAllowed to
         ensure that it does not violate security.

         The evaluation of that expression takes place under the

       security credentials of the creator of its expExpressionEntry.

       To maintain security of MIB information, expression evaluation must
       take place using security credentials for the implied Gets of the
       objects in the expression as inputs (conceptually) to
       isAccessAllowed from the Architecture for Describing SNMP
       Management Frameworks. These are the security credentials of the
       creator of the corresponding expExpressionEntry."
    INDEX   { expExpressionOwner, expExpressionName,
                   IMPLIED expValueInstance }
    ::= { expValueTable 1 }

ExpValueEntry ::= SEQUENCE {
    expValueInstance   OBJECT IDENTIFIER,
    expValueCounter32Val   Counter32,
    expValueUnsigned32Val   Unsigned32,
    expValueTimeTicksVal   TimeTicks,
    expValueInteger32Val   Integer32,
    expValueIpAddressVal   IpAddress,
    expValueOctetStringVal   OCTET STRING,
    expValueOidVal   OBJECT IDENTIFIER,
    expValueCounter64Val   Counter64
}

expValueInstance OBJECT-TYPE
    SYNTAX   OBJECT IDENTIFIER
    MAX-ACCESS not-accessible
    STATUS   current
    DESCRIPTION
     "The final instance portion of a value's OID according to
      the wildcarding in instances of expObjectID for the
      expression. The prefix of this OID fragment is 0.0,
      leading to the following behavior.

      If there is no wildcarding, the value is 0.0.0. In other
      words, there is one value which standing alone would have
      been a scalar with a 0 at the end of its OID.

      If there is wildcarding, the value is 0.0 followed by
      a value that the wildcard can take, thus defining one value
      instance for each real, possible value of the wildcard.
      So, for example, if the wildcard worked out to be an ifIndex,
      there is an expValueInstance for each applicable ifIndex."
    ::= { expValueEntry 1 }

expValueCounter32Val OBJECT-TYPE
    SYNTAX   Counter32
    MAX-ACCESS read-only

```
    STATUS        current
    DESCRIPTION
     "The value when expExpressionValueType is 'counter32'."
    ::= { expValueEntry 2 }


expValueUnsigned32Val OBJECT-TYPE
    SYNTAX        Unsigned32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
     "The value when expExpressionValueType is 'unsigned32'."
    ::= { expValueEntry 3 }


expValueTimeTicksVal OBJECT-TYPE
    SYNTAX        TimeTicks
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
     "The value when expExpressionValueType is 'timeTicks'."
    ::= { expValueEntry 4 }


expValueInteger32Val OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
     "The value when expExpressionValueType is 'integer32'."
    ::= { expValueEntry 5 }


expValueIpAddressVal OBJECT-TYPE
    SYNTAX        IpAddress
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
     "The value when expExpressionValueType is 'ipAddress'."
    ::= { expValueEntry 6 }


expValueOctetStringVal OBJECT-TYPE
    SYNTAX        OCTET STRING (SIZE (0..65536))
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
     "The value when expExpressionValueType is 'octetString'."
    ::= { expValueEntry 7 }


expValueOidVal OBJECT-TYPE
    SYNTAX        OBJECT IDENTIFIER
    MAX-ACCESS    read-only
```

    STATUS   current
    DESCRIPTION
     "The value when expExpressionValueType is 'objectId'."
    ::= { expValueEntry 8 }

expValueCounter64Val OBJECT-TYPE
    SYNTAX   Counter64
    MAX-ACCESS   read-only
    STATUS   current
    DESCRIPTION
     "The value when expExpressionValueType is 'counter64'."
    ::= { expValueEntry 9 }

--
-- Conformance
--

dismanExpressionMIBConformance OBJECT IDENTIFIER ::=
    { dismanExpressionMIB 3 }
dismanExpressionMIBCompliances OBJECT IDENTIFIER ::=
    { dismanExpressionMIBConformance 1 }
dismanExpressionMIBGroups   OBJECT IDENTIFIER ::=
    { dismanExpressionMIBConformance 2 }

-- Compliance

dismanExpressionMIBCompliance MODULE-COMPLIANCE
     STATUS current
     DESCRIPTION
         "The compliance statement for entities which implement
         the Expression MIB."
     MODULE   -- this module
         MANDATORY-GROUPS {
             dismanExpressionResourceGroup,
             dismanExpressionDefinitionGroup,
             dismanExpressionValueGroup
         }

    OBJECT   expResourceDeltaMinimum
    SYNTAX   Integer32 (-1 | 60..600)
    DESCRIPTION
         "Implementation need not allow deltas or it may
         implement them and restrict them to higher values."

    OBJECT   expObjectSampleType
    WRITE-SYNTAX   INTEGER { absoluteValue(1) }
    DESCRIPTION
         "Implementation may disallow deltas calculation or

```
        change detection."

     OBJECT          expObjectIDWildcard
     WRITE-SYNTAX    INTEGER { false(2) }
     DESCRIPTION
          "Implementation may allow wildcards."

     OBJECT          expObjectDiscontinuityIDWildcard
     WRITE-SYNTAX    INTEGER { false(2) }
     DESCRIPTION
          "Implementation need not allow wildcards."

     OBJECT           expObjectConditionalWildcard
     WRITE-SYNTAX    INTEGER { false(2) }
     DESCRIPTION
          "Implementation need not allow deltas wildcards."

     ::= { dismanExpressionMIBCompliances 1 }

-- Units of Conformance

dismanExpressionResourceGroup OBJECT-GROUP
     OBJECTS {
          expResourceDeltaMinimum,
          expResourceDeltaWildcardInstanceMaximum,
          expResourceDeltaWildcardInstances,
          expResourceDeltaWildcardInstancesHigh,
          expResourceDeltaWildcardInstanceResourceLacks
     }
     STATUS current
     DESCRIPTION
          "Expression definition resource management."
     ::= { dismanExpressionMIBGroups 1 }

dismanExpressionDefinitionGroup OBJECT-GROUP
     OBJECTS {
          expExpression,
          expExpressionValueType,
          expExpressionComment,
          expExpressionDeltaInterval,
          expExpressionPrefix,
          expExpressionErrors,
          expExpressionEntryStatus,

          expErrorTime,
          expErrorIndex,
          expErrorCode,
          expErrorInstance,
```

```
            expObjectID,
            expObjectIDWildcard,
            expObjectSampleType,
            expObjectDeltaDiscontinuityID,
            expObjectDiscontinuityIDWildcard,
            expObjectDiscontinuityIDType,
            expObjectConditional,
            expObjectConditionalWildcard,
            expObjectEntryStatus
    }
    STATUS current
    DESCRIPTION
        "Expression definition."
    ::= { dismanExpressionMIBGroups 2 }

dismanExpressionValueGroup OBJECT-GROUP
    OBJECTS {
            expValueCounter32Val,
            expValueUnsigned32Val,
            expValueTimeTicksVal,
            expValueInteger32Val,
            expValueIpAddressVal,
            expValueOctetStringVal,
            expValueOidVal,
            expValueCounter64Val
    }
    STATUS current
    DESCRIPTION
        "Expression value."
    ::= { dismanExpressionMIBGroups 3 }

END
```

4. 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.

5. Acknowledgements

   This MIB contains considerable contributions from the Distributed
   Management Design Team (Andy Bierman, Maria Greene, Bob Stewart, and
   Steve Waldbusser), and colleagues at Cisco who did the first
   implementation.

6. References

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

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

   [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.

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

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

   [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.

   [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.

   [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.

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

   [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.

   [RFC1903]   Case, J., McCloghrie, K., Rose, M. and S. Waldbusser,
                "Coexistence between Version 1 and version 2 of the
                Internet-standard Network Management Framework", RFC
                1903, January 1996.

   [RFC2981]   Stewart, B., "Event MIB", RFC 2981, October 2000.

   [PracPersp] Leinwand, A. and K. Fang, "Network Management: A
                Practical Perspective", Addison-Wesley Publishing
                Company, Inc., 1993.

7. Security Considerations

   Expression MIB security involves two perspectives: protection of
   expressions from tampering or unauthorized use of resources, and
   protection of the objects used to calculate the expressions.

Security of expression definitions and results depends on the
expression owner (expExpressionOwner). With view-based access
control [RFC2575] a network manager can control who has what level of
access to what expressions.

Access control for the objects within the expression depends on the
security credentials of the expression creator. These are the
security credentials used to get the objects necessary to evaluate
the expression. They are the security credentials that were used to
set the expExpressionRowStatus object for that expression to
'active', as recorded by the managed system.

This means that the results of an expression could potentially be
made available to someone who does not have access to the raw data
that went into them. This could be either legitimate or a security
violation, depending on the specific situation and security policy.

To facilitate the provisioning of access control by a security
administrator for this MIB itself using the View-Based Access Control
Model (VACM) defined in RFC 2575 [RFC2575] for tables in which
multiple users may need to independently create or modify entries,
the initial index is used as an "owner index". Such an initial index
has a syntax of SnmpAdminString, and can thus be trivially mapped to
a securityName or groupName as defined in VACM, in accordance with a
security policy.

All entries in related tables belonging to a particular user will
have the same value for this initial index. For a given user's
entries in a particular table, the object identifiers for the
information in these entries will have the same subidentifiers
(except for the "column" subidentifier) up to the end of the encoded
owner index. To configure VACM to permit access to this portion of
the table, one would create vacmViewTreeFamilyTable entries with the
value of vacmViewTreeFamilySubtree including the owner index portion,
and vacmViewTreeFamilyMask "wildcarding" the column subidentifier.
More elaborate configurations are possible.

8. Author's Address

    Bob Stewart
    Cisco Systems, Inc.
    170 West Tasman Drive
    San Jose, CA 95134-1706
    U.S.A.

9. Editor's Address

    Ramanathan Kavasseri
    Cisco Systems, Inc.
    170 West Tasman Drive
    San Jose, CA 95134-1706
    U.S.A.

    Phone: +1 408 527 2446
    EMail: ramk@cisco.com

10. Full Copyright Statement

Acknowledgement