Desktop Management Task Force

DMI to SNMP Mapping Standard

DSP0002

Version 1.0 – November 25, 1997

Approved by the Technical Advisory Committee on December 9, 1997

Approved by the Steering Committee on December 10, 1997

Technical inquiries and editorial comments should be directed in writing to:

Desktop Management Task Force (DMTF)
M/S JF2-53
2111 N.E. 25th Avenue
Hillsboro, OR 97124
PHONE: (503) 264-9300
FAX: (503) 264-9027
email: **dmtf-info@dmtf.org**

Additional electronic copies of this specification can be obtained free of charge from the Internet at:

ftp://ftp.dmtf.org

or

from the World Wide Web at:

http://www.dmtf.org

Additional hardcopies can be obtained for a fee by contacting the DMTF at the address listed above.

IMPORTANT INFORMATION AND DISCLAIMERS

1. THIS SPECIFICATION (WHICH SHALL INCORPORATE ANY REVISIONS, UPDATES, AND MODIFICATIONS HERETO) IS FURNISHED FOR INFORMATIONAL PURPOSES ONLY. COMPAQ COMPUTER CORPORATION, DELL COMPUTER CORPORATION, DIGITAL EQUIPMENT CORPORATION, HEWLETT-PACKARD COMPANY, INTEL CORPORATION, INTERNATIONAL BUSINESS MACHINES CORPORATION, MICROSOFT CORPORATION, NEC TECHNOLOGIES, INC., NOVELL INC., THE SANTA CRUZ OPERATION, SUN MICROSYSTEMS, INC., SYMANTEC, OR ANY OTHER DMTF MEMBER MAKE NO WARRANTIES WITH REGARD THERETO, AND IN PARTICULAR DO NOT WARRANT OR REPRESENT THAT THIS SPECIFICATION OR ANY PRODUCTS MADE IN CONFORMANCE WITH IT WILL WORK IN THE INTENDED MANNER OR BE COMPATIBLE WITH OTHER PRODUCTS IN NETWORK SYSTEMS. NOR DO THEY ASSUME RESPONSIBILITY FOR ANY ERRORS THAT THE SPECIFICATION MAY CONTAIN OR HAVE ANY LIABILITIES OR OBLIGATIONS FOR DAMAGES INCLUDING, BUT NOT LIMITED TO, SPECIAL, INCIDENTAL, INDIRECT, PUNITIVE, OR CONSEQUENTIAL DAMAGES WHETHER ARISING FROM OR IN CONNECTION WITH THE USE OF THIS SPECIFICATION IN ANY WAY. CORPORATIONS MAY FOLLOW OR DEVIATE FROM THIS SPECIFICATION AT ANY TIME.

2. NO REPRESENTATIONS OR WARRANTIES ARE MADE THAT ANY PRODUCT BASED IN WHOLE OR IN PART ON THE ABOVE SPECIFICATION WILL BE FREE FROM DEFECTS OR SAFE FOR USE FOR ITS INTENDED PURPOSE. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT DOES SO AT HIS OWN RISK.

3. THE USER OF THIS SPECIFICATION HEREBY EXPRESSLY ACKNOWLEDGES THAT THE SPECIFICATION IS PROVIDED AS IS, AND THAT THE DMTF, NEITHER INDIVIDUALLY NOR COLLECTIVELY, MAKE ANY REPRESENTATIONS, EXTEND ANY WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, ORAL OR WRITTEN, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTY OR REPRESENTATION THAT THE SPECIFICATION OR ANY PRODUCT OR TECHNOLOGY UTILIZING ANY ASPECT OF THE SPECIFICATION WILL BE FREE FROM ANY CLAIMS OF INFRINGEMENT OF INTELLECTUAL PROPERTY, INCLUDING PATENTS, COPYRIGHT AND TRADE SECRETS OF ANY THIRD PARTY, OR ASSUMES ANY OTHER RESPONSIBILITIES WHATSOEVER WITH RESPECT TO THE SPECIFICATION OR SUCH PRODUCTS. IN NO EVENT WILL DMTF MEMBERS BE LIABLE FOR ANY LOSSES, DAMAGES INCLUDING, WITHOUT LIMITATION, THOSE DAMAGES DESCRIBED IN SECTION 1 ABOVE, COSTS, JUDGMENTS, OR EXPENSES ARISING FROM THE USE OR LICENSING OF THE SPECIFICATION HEREUNDER.

Table of Contents

Revision History

Original Proposal: July 7, 1995 (Steve Bostock)

Version 1.0: November 25, 1997 (Brian O'Keefe)

- Updated to use final DMI 2.0 and SNMPv2 conventions and methods, including formal DMI pragma statement.
- Added new section for specifying DMI to SNMP Row Creation/Deletion.
- Added new sections specifying DMI Event to SNMP Notification mapping.
- Added additional SNMP Pragmas for new DMTF standard classes
- Use standard DMTF cover page and disclaimer.

1 Introduction

This document specifies a Desktop Management Task Force (DMTF) standard. It defines a set of mapping procedures to enable systems instrumented to the Desktop Management Interface (DMI) to be remotely, and uniformly, managed using the Simple Network Management Protocol (SNMP).

The Desktop Management Framework and the Internet-standard Network Management Framework, commonly known as the "SNMP Management Framework", are standard management frameworks widely deployed to manage computer systems and network devices, respectively. The two frameworks are similar in concept and function. However, while the two frameworks may co-exist on the same system, the two are not inherently interoperable. Despite this, applications that span the heterogeneous nature of system and network management must access management information using both frameworks. Therefore, the objective of this mapping standard is to bridge the interoperability gap between SNMP and DMI-based solutions. For example, by providing the mapping specified in this document, existing SNMP-based applications and/or toolkits can be leveraged to manage DMI-based systems as well as SNMP devices.

There are three distinct facets to the mapping, and thus interoperation, of SNMP- and DMI-based management solutions:

1. to enable SNMP applications to manage DMI instrumented systems,
2. to enable DMI instrumented systems to support standard MIBs, and
3. to provide SNMP access to DMI-instrumented systems that may or may not have a local SNMP mapping agent resident.

A final consideration made in this specification is distinction between the two basic models of management provided for in both frameworks: directed management information access and unsolicited, event-triggered notifications. As such, the mappings specified in this document provide flexibility to separate or unify the mapping implementation for management access versus event-triggered notifications. For example, a partial solution may only implement the notification mappings.

1.1 Enabling SNMP Management Applications

As new MIFs are defined, whether standard or proprietary, new management applications will need to be written to reap the benefits of this new instrumentation. Management applications span the spectrum from simple "MIB Browsers" that understand only the syntax of the managed objects - enough to display their values as text, graphs, tables, etc.; through to "expert" applications that fully understand the semantics of the managed objects and can perform complex management tasks autonomously.

Developing management applications, particularly "expert" ones, is time consuming and expensive. For such applications to be commercially viable, they need a large potential market. Standard instrumentation with a consistent behavior across a wide variety of components and platforms is a major factor in creating this large market. The solution presented in this document was driven by this requirement; specifically, it facilitates:

a. writing a standard SNMP-based management application that can manage the functionality defined by a set of standard DMTF MIF groups, independent of both the component that implements those groups, and the system on which that component is installed.

b. writing a single, general, DMI to SNMP mapping agent for any given platform that will perform these mappings and provide SNMP access to any component MIF that may get installed on that system.

c. consistent behavior and full interoperability between implementations.

1.2 Enabling Support of Standard MIBs

There are now thousands of standard MIB objects defined in the areas of network management, system management, and application management. There are also many, widely deployed, SNMP management applications that depend on these standard objects.

Although DMI-based instrumentation can coexist on the same platform with non-DMI instrumentation, there is a growing desire to instrument computer systems solely to the DMI, but still be able to support standard SNMP MIBs and their corresponding management applications. For this to succeed, it must be transparent to an SNMP management application whether the MIB is implemented using DMI instrumentation or some other scheme.

Unfortunately, the combination of technical incompatibilities between the DMI and SNMP frameworks and cultural differences between the DMTF and IETF organizations, preclude a general, automated, solution to this problem.

Major incompatibilities between the two frameworks include the following:

- The DMI 1.x has no defined mechanism to add or delete rows in a table. The DMI 2.0 now provides these operations. SNMPv2 also defines a formal convention for adding and deleting rows in a table. However, these methods differ between DMI and SNMP.

- The DMI has no defined mechanism to permit interaction between tables (there is no equivalent of an OBJECT IDENTIFIER that can be used as a generalized pointer).

- The DMI does not support OBJECT IDENTIFIERs as data types.

- The DMI does not support scalar variables, all variables are (potentially) multi-instanced.

An automated solution requires that the DMI instrumentation support those objects required by the IETF standard MIBs; or, at least, a set of objects from which they can be derived. By and large, the availability of such objects is not a technical problem, but rather, a political and cultural one. Standard MIFs (like MIBs) are designed by committee and are shaped as much by politics and personalities as technical need. To make progress on this issue, DMTF working committees must recognize the importance of supporting standard MIBs and make the effort to define the necessary attributes while resisting the urge to make gratuitous omissions and semantic changes.

In practice, each standard MIB to be supported will probably require a custom coded SNMP sub-agent. At worst, this is no different from writing an SNMP sub-agent to implement a standard MIB on any system (instrumented to the DMI or not). At best, the SNMP sub-agent code can be reduced to a very thin layer (perhaps largely table driven) if there are standard DMTF MIFs that support the same set of managed objects (attributes) as the standard IETF MIBS.

Note that even though a custom coded SNMP sub-agent may be required for each MIB, the DMI still provides the benefit of a standard, platform independent, interface to the instrumentation. Any sub-agent code could be written to be largely portable between systems, only the interfaces to the SNMP master agent and host OS would need to be recoded between systems.

1.3 Enabling Support for Non-resident Mapping Agents

Many computer systems that are DMI instrumented do not have an SNMP Agent installed and running on that system. Therefore, in addition to providing the specifications for an SNMP-to-DMI mapping agent resident on the managed system, the mapping agent must also be able to run on a different system, acting as a proxy, that remotely accesses the DMI instrumentation using one of the DMI 2.0 RPCs.

By convention, an SNMP-to-DMI mapping agent that acts in a proxy role for remote DMI-based systems should use the SNMP community string (SNMPv1 and SNMPv2C) to determine the target DMI system. The format of the SNMP community string for this purpose is a URL-like string of the form:

dmi://target-address[/access-control-token]

where, "target-address" is the network-layer address (e.g., IP or IPX) of the target DMI system, and "access-control-token" is an optional string that conforms to the, yet to be defined, DMTF conventions for security and access control. It is an implementation specific matter for the SNMP-to-DMI mapping agent to determine and select which RPC to use for remote DMI access.

2 Overview

2.1 The Desktop Management Interface

The Desktop Management Interface (DMI) is defined by the Desktop Management Task Force (DMTF). It provides a management framework that consists of three elements:

- a Management Information Format (MIF), which specifies a structure and format for defining manageable attributes;
- a set of standard MIF definitions for hardware and software components; and,
- a standard Application Programming Interface (API) for local access to the management information.

The term "DMI" is used, interchangeably, to refer to either the framework or the API.

The DMI API actually consists of two separate APIs: the Management Interface (MI), through which management applications interact with managed objects; and the Component Interface (CI), through which managed components provide access to dynamic data and generate indications. The DMI also defines an active, resident, piece of code termed the Service Provider (SP) which mediates between the MI and CI and performs services on behalf of each.

The DMI 1.x is a local interface, to be used within a single system. The DMI is not designed to replace existing network management protocols. It is designed to provide a consistent method for providing instrumentation to those protocols. The Service Provider is the broker of local instrumentation.

The DMI 2.0 adds a remote interface using any of three specified Remote Procedure Protocol (RPC) options. The Management Interface (MI) API is also changed to be procedure based rather than stream based. Finally, DMI 2.0 adds operations for explicit row creation and removal; plus standard configuration tables for remotely subscribing for DMI Indications and Events.

In the DMI, "components" are physical or logical entities on a (computer) system, such as hardware, software or firmware. Components generally correspond to products, and may come with the system or may be added to it. Components have one or more named "attributes" that collectively define the information available to a management application. Attributes are collected into named "groups" for ease of reference. Groups may be scalar or may be replicated. Replicated groups are called "tables", and a "row" (instance) of a table is referred to by a set of attributes that form a "key".

So, within a system, there are many components, each with one or more groups. Each group has one or more attributes; and each group may be replicated as a table. The Service Provider presents this component/group/attribute/key representation to the management application.

In the DMI, standardization occurs on the group level. Each group definition, or class, is named with a globally unique class-string that, by convention, has the form:

"defining-body|specific-name|version"

In this convention, defining body is the name of the organization (such as "DMTF", "IEEE", "Acme Computer", etc.) defining the group; specific name identifies the contents of the group ("FRU", "System Resources", etc.) And version identifies the

version of the group definition ("001", "002", etc.).

A standard MIF such as the "PC Systems Standard MIF Definition" [17] is actually just a collection of standard groups that are relevant to instrument some hardware or software component - in this case a desktop PC chassis.

The DMTF model is that a component typically corresponds to a physical product; for example, an adapter board and its associated driver(s), a shrink-wrapped software program/package, a PC-compatible computer, etc. The component vendor creates a MIF file that describes the manageable characteristics of that component; the component MIF may include groups from one or more "standard MIFs" and vendor proprietary MIFs as applicable. For example a desktop PC with an Ethernet adapter integrated on the mother board may ship with a single MIF that includes groups from both the "PC Systems Standard MIF Definition" [17] plus groups from the "LAN Adapter Standard Groups Definition" [15].

2.2 The Simple Network Management Protocol

The Internet-standard Network Management Framework is defined by the Internet Engineering Task Force (IETF). It too consists of three elements:

- the Structure and identification of Management Information (SMI) that specifies how to define managed object;
- a Management Information Base (MIB), which is a set of standard managed objects for Internet devices; and
- the Simple Network Management Protocol (SNMP), which defines the protocol used to manage these objects.

The term "SNMP" generally refers to both the Internet-standard Network Management Framework and the protocol component of that framework. Two versions of the Internet-standard Network Management Framework exist today. Version 1 (SNMPv1) [1-3] is a full Internet standard and has become the de-facto standard for remote management within the computer industry. Version 2 (SNMPv2) [4-11] further enhances the SNMP Structure of Management Information, plus adds new protocol operations for bulk retrieval and acknowledged notification. Version 3 (SNMPv3) is currently in process of being defined by the IETF to add secure management via authentication and privacy extensions.

An SNMP management application, as opposed to a simple browser, needs a priori knowledge of the names and semantics of the objects it is designed to manage. In the SNMP, an Object Identifier (OID) provides the unique and authoritative identification (name) for each managed object (aka, attribute). An SNMP MIB provides these names and semantics for some cohesive subset of the manageable features of a device class. The view presented by the MIB is constant irrespective of the vendor or the physical packaging of the device. For example, an application designed to manage TCP/IP according to MIB-II will work for any (conformant) TCP/IP implementation; whether it is part of NetWare, Windows95, a Cisco router, or a toaster.

2.3 Elements of the Mapping Solution

A collection of related DMTF group classes is semantically equivalent to an SNMP MIB, in that it models specific functionality independent of physical packaging. The principle difference is the way in which DMI and SNMP management information is named -- DMI uses class names, SNMP uses numeric object identifiers. The mapping scheme described in this document is based on this observation. The scheme allows (DMTF) standard MIBs to be generated from standard MIFs, and for proprietary MIBs to

be generated from proprietary MIFs. These MIBs can then be used by an SNMP-based application in conjunction with a mapping agent to manage any DMI-instrumented system whose MIF is made up of these groups. For example, the LAN Adapter MIB derived from the "LAN Adapter Standard Groups Definition" MIF can be used to manage the adapter whether it is an integrated part of the mother board component, or a separately purchased plug in card.

The solution defined in this document has the following elements:

- An administrative procedure for assigning SNMP Object Identifiers to MIF group classes in such a way that related groups can be automatically grouped together into cohesive MIB modules. A method of embedding OIDs in the MIF file such that the pairing of class string and OID is available at run-time through the MI

- A set of algorithms for converting MIF attributes into SNMP objects, such that a general mapping agent can be written to perform the reverse algorithm at run time using only information available through the MI. A MIF-to-MIB conversion program that inputs a MIF file and outputs one or more MIB files according to the translation scheme defined in this document.

- A standard DMTF DMI MIB that provides access to DMI meta-data such as names, types, enumerations, descriptions, etc.; for all installed MIFs on a system. An SNMP application could upload the information from this MIB and generate the same MIB modules as the MIF-to-MIB program. The DMI MIB also provides the "containment" information lost in the mapping scheme: a list of all components installed on the system, and a list of all groups implemented by each component.

- A general DMI to SNMP mapping agent that resides on the managed node and acts as an SNMP sub-agent to the SNMP master agent on that node. The mapping agent will register for all Object Identifier (OID) subtrees supported by registered DMI components and will satisfy incoming SNMP operations on those OIDs by making appropriate calls to the MI. The mapping agent will conform to the mappings and guidelines defined in this document. The mapping agent will also implement the DMI MIB referred to above in its entirety. Note, by using DMI 2.0 remote capabilities, the mapping agent could alternatively reside on another system acting as a proxy for the DMI-instrumented system.

3 OID Assignment Procedure

One of the most fundamental differences between the DMI and SNMP frameworks is how management information is identified or named. To bridge the cap between the two frameworks, a mapping of DMI class name to SNMP object identifier must be defined.

Two methods for assigning an ASN.1 Object Identifier (OID) to a DMI Class are defined by this specification. The first, and recommended, method is administrative assignment of OIDs to each DMI Class defined by the DMTF or other organization or company. However, in the case where an OID has not been administratively assigned, an automated procedure for generating deterministic and unique OIDs for a DMI Class is also provided.

3.1 Administratively Assigned OIDs

The following administrative procedure shall be followed by the DMTF when producing a standard MIF document. A similar procedure should be followed by other organizations and companies who define proprietary MIFs, except that the object identifier (OID) for the MIF should be allocated under their respective enterprise node.

Assign a unique OID for each set of related groups in the MIF under the branch:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1)
 dmtf(412) dmtfStdMifs(2) }
```

A standard MIF will normally be cohesive and only include a single set of related groups, thus all groups in the MIF will be rooted under a single OID sub-tree and will be grouped together into a single SNMP MIB module. As of this writing, the Technical Advisory Committee Chair is the assignment authority for new OIDs immediately under the dmtfStdMifs(2) branch. If additional groups are added to an existing related set, the same intermediate OID branch should be used for the new groups as for the existing related groups.

For each non-event group class definition in the MIF assign it a unique OID directly under MIF OID, such that there is a one-to-one correspondence between group class-strings and OIDs. For each event group class definition in the MIF assign it the same OID as the associated group class definition, provided there is a one-to-one correspondance. Proprietary event group extensions must assign a unique OID so as not to conflict with the standard event group.

The SNMP OID for each DMI Class is assigned using the "pragma" statement defined by the DMI 2.0 MIF Grammar. Each DMI Class (group) definition should include a pragma of the form:

```
pragma = "SNMP:<OID> ;"
```

Where <OID> is replaced by the actual SNMP OID assigned.

OID assignments for DMTF standard MIFs are defined in the MASTER.MIF. Section 5.1.1 of this document lists those assignments for standard MIF groups defined prior to the adoption of this DMI-to-SNMP mapping standard.

Note: if the class-string obeys the naming conventions and has a numeric "version" string, the same OID shall be used for all versions of the group (later versions are backwards compatible).

3.2 Dynamically Generated OIDs

For each group class that does not have an administratively assigned SNMP OID mapping, a dynamically generated OID may be assigned. If dynamic OID generation is supported by a mapping agent, the following procedure shall be used to generate a unique OID under the branch:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1)
  dmtf(412) dmtfDynOids(3) }
```

The generated OID will be predictable in the sense that any two implementations, given the same group class string, will generate identical OIDs. The generated OID will also have a very high probability of being globally unique.

This scheme will allow all DMI data to be accessed via SNMP even if no a priori administrative OID assignments were made. An SNMP management application can discover the OID assignments, syntax, access, descriptions, etc. through the DMTF-DMI-MIB. A MIB browser type application could be constructed using these OIDs and would have a high probability of working across all machines with those MIFs installed.

The OID is generated from the group class string using the MD5 digest algorithm. This algorithm is shamelessly borrowed from the SNMPv2 Simplified Configuration Model Internet Draft.

Modify the group class string, if necessary, as follows: if the string follows the convention and includes the numeric version number field, remove the version number by truncating immediately following the final '|'.

Form a string of 1,048,576 octets by repeating the value of the modified class string as often as necessary, truncating accordingly, and use the resulting string as input to the MD5 algorithm. The resulting 16-octet digest is then treated as four unsigned 32-bit integers (A, B, C, D) in network-byte order:

```
A0 A1 A2 A3 B0 B1 B2 B3 C0 C1 C2 C3 D0 D1 D2 D3
where x0 is most significant byte, x3 is least significant
```

The SNMP OID for the corresponding DMI Class is formed by concatenating the above OID prefix with the 4 subids just generated:

```
1.3.6.1.4.1.412.3.A.B.C.D
```

Implementor’s Note: the generation of a 1 Mb string of octets described above need not be implemented as a full 1 Mb block of data space. Most public domain implementations of the MD5 algorithm provide a means to iterate through a conceptually long block of input octets by repeatedly calling the MD5 functions on shorter segments of the conceptual string.

4 MIF to MIB Conversion

MIF to MIB conversion may be performed by hand, but would normally be performed using an automated conversion program. In either case, the following algorithm shall be used.

Inspect each group definition in the MIF and retrieve its OID from the SNMP pragma included in the group definition. Split each OID into a trailing sub-id and a prefix. A separate MIB module will be output for each distinct prefix. All groups with the same prefix will be part of the same MIB module.

Groups with no SNMP pragma definition may be converted by using the algorithm described in section 3.2 to dynamically generate an SNMP OID for the group. All groups with dynamically generated SNMP OIDs can be output to the same MIB module or separate MIB modules (e.g., grouped according to user intervention and selection).

For each group definition in the MIF:

> If the class string has the form "EventGeneration|*|*" generate a series of notification (or trap) definitions. See section "Event Mapping" below.
>
> Otherwise, generate an SNMP conceptual table. See section "Managed Object Mapping below.

For each global enumeration defined in the MIF, generate an equivalent SNMP textual convention. Apply the name mappings (see below) to both the name and the enumeration literals. Output the textual convention in each MIB module that has a reference to it.

4.1 Name Mappings

An SNMP/ASN.1 "name" (formally called a "descriptor") is a string of at least one character, drawn from A-Z, a-z, 0-9, and hyphen. In a name, a hyphen cannot appear as the last character or adjacent to another hyphen. Names are case-sensitive, and the first character of a name must be a letter. Identifiers must start with a lower-case letter. Type and module names must start with an upper-case letter. All types and variables defined in a single module must have unique names. SNMPv2 is more restrictive, identifiers may not include hyphens and must not exceed 64 characters in length.

DMI names can be any sequence of characters (up to 255) drawn from either the ISO 8859-1 or Unicode character set (but not both). C-style escape characters are also allowed.

4.1.1 Name Mapping Algorithm

The following general algorithm is used to convert a DMI name to a legal SNMPv2 name. Name mappings for each specific type of SNMP name also involve additional procedures described in subsequent sections.

1. For each sequence of illegal characters and hyphens, discard the entire sequence and convert the next character (if a letter) to upper-case. If the first character of the resulting string is not alphabetic, prepend the character 'x'.

2. If the name being converted is a DMI Class name, truncate the resulting string to 59 characters, which leaves room for the various suffixes appended below;

otherwise, trunctate the name to 64 characters.

3. If the name refers to a type (used in an SNMP SYNTAX statement), convert the first character to upper-case; otherwise, convert the first character to lower-case.

4. If the name is not unique within the MIB module, append an ordinal (truncate if necessary) such that the name becomes unique. Note, names given to numbers within an enumeration list are only required to be unique within that particular list.

4.1.2 Summary of Name Mappings

The elements of procedure described in subsequent sections include additional procedures for mapping DMI names, using the general algorithm above, to produce the complete SNMP name for a particular usage. This section summarizes those name mappings for convenience and quick reference.

<className> refers to the name mapping, per the algorithm above, of the specific-name portion of the DMI Class string for non-Event Generation Groups.

<assocName> refers to the specific-name-of-assoc-group portion of DMI Event Generation class strings. Note, this is also is equivalent to the specific-name portion of the DMI Associated Group attribute value.

<eventType> refers to the name mapping of the Event Type enumeration literal for DMI Event Generation classes.

<enumName> refers to the name mapping of the name clause in a DMI ENUM definition.

<enumLiteral> refers to the name mapping of the DMI Enumeration string literals in a DMI ENUM definition.

SNMP Descriptor for	Mapping Convention	Description
Table OBJECT	<className>Table	Append "Table" to <className>
Table Entry OBJECT	<className>Entry	Append "Entry" to <className>
Table Entry SYNTAX	<ClassName>Entry	Append "Entry" to <className>, first character is upper-case
RowStatus OBJECT	<className>State	Append "State" to <className>
Attribute OBJECTs	<attributeName>	
Event Parent OID	<assocName>Traps	Append "Traps" to <className>
Event NOTIFICATION	<assocName><eventType> or <assocName>Ev<###>	Append <eventType> name to <assocName>; or if result too long, append "Evt" and the Event Type value to <assocName>
Event System	<assocName>EvSys	Append "EvSys" to <assocName>
Event Subsystem	<assocName>EvSub	Append "EvSub" to <assocName>
Event Solution	<assocName>EvSol	Append "EvSol" to <assocName>
Textual Convention	<enumName>	
Enum literal	<enumLiteral>	
Intermediate level Dynamic OIDs	dmiDynOid<#>	Append sub-id value to "dmiDynOid"

4.2 Managed Object Mapping

Each non-event group class becomes an SNMP conceptual table. The OID of the conceptual table object is the OID assigned to the group class {grpOID}.

The conceptual row within the table object is assigned the sub-id "1".

OID = {grpOID.1}

Each attribute within the DMI group becomes a columnar object of that table, with the attribute id being used as a sub-id.

OID = {grpOID.1.attrId}

4.2.1 Mapping to the OBJECT-TYPE <descriptor>

For columnar objects, the descriptor is formed by applying the name mapping algorithm defined above on the value of the MIF Name statement.

For the conceptual table object, the descriptor is formed by applying the name mapping algorithm defined above on the "specific name" portion of the class string and appending the text "Table".

For the conceptual row object, the descriptor is formed by applying the name mapping algorithm defined above on the "specific name" portion of the class string and appending the text "Entry".

4.2.2 Mapping to the SYNTAX clause

For columnar objects, the table below shows the mapping of data types from DMI to SNMPv1 and SNMPv2 (where different). Areas where the two frameworks mismatch are flagged and discussed in the notes below.

DMI	SNMPv1	SNMPv2
integer	INTEGER	Integer32
integer64 *1	OCTET STRING (SIZE(8))	
gauge	Gauge	Gauge32
counter	Counter	Counter32
counter64 *2	Counter	Counter64
string(n) *3, displaystring(n)	OCTET STRING(SIZE(0..n))	
octetstring(n)	OCTET STRING(SIZE(0..n))	
date *4	OCTET STRING(SIZE(25))	

Notes:

1. SNMP has no Integer64 data type. Integer64 will be represented by a string of exactly 8 octets in network byte order. The DMTF-DMI-MIB defines an appropriate textual convention "DmiInteger64".

2. SNMPv1 has no Counter64 data type. Counter64 will be represented in SNMPv1 as a Counter by taking the least significant 32 bits.

3. The SNMP textual convention "DisplayString" represents textual information taken from the NVT ASCII character set, as defined in pages 4, 10-11 of RFC 854. This definition is too restrictive to handle DMI strings and displaystrings, which can be either ISO8859-1 or Unicode; hence DMI strings (displaystring) are represented as a sequence of octets. The DMTF-DMI-MIB defines an appropriate textual convention "DmiString".

4. Date is represented by a DisplayString of exactly 25 octets. The string has the same format as the DMI date except that the trailing three octets of '\0's are omitted. The DMTF-DMI-MIB defines an appropriate textual convention "DmiDate".

For the conceptual table and row objects, the value of the SYNTAX clause is SEQUENCE OF <x> and <x> respectively, where <x> is the result of applying the name mapping algorithm defined above on the "specific name" portion of the class string.

4.2.3 Mapping to the ACCESS or MAX-ACCESS clause

Mapping the DMI access definition to the SNMP ACCESS (SNMPv1 SMI) or MAX-ACCESS (SNMPv2 SMI) clause varies according to the following three conditions:

1. The attribute is a "key" attribute and hence appears in the INDEX clause.

   The ACCESS clause is set to "read-only" for SNMPv1, and the MAX-ACCESS clause is set to "not-accessible" for SNMPv2; unless there would be no accessible columnar objects, in which case it is set to "read-only".

2. The attribute is not a key attribute; and either no key attributes exist for the class, or all of the key attributes have DMI access equal to read-only.

   In this case, rows of the table may not be created through the management interface. Therefore, the SNMP ACCESS or MAX-ACCESS clause is mapped as follows:

DMI	SNMPv1	SNMPv2
read-only	read-only	read-only
read-write	read-write	read-write
write-only	write-only	read-write

3. The attribute is not a key attribute; and one or more key attributes for the class have DMI access equal to read-write or write-only.

   In this case, rows of the table may be created through the management interface. Therefore, the SNMP ACCESS or MAX-ACCESS clause is mapped as follows and the procedure described in section 4.2.8 is followed to include a RowStatus columnar object in the resulting SNMP table.

DMI	SNMPv1	SNMPv2
read-only	read-only	read-only
read-write	read-write	read-create
write-only	write-only	read-create

4.2.4 Mapping to the STATUS clause

The value of the STATUS clause will always be "mandatory" for SNMPv1, and "current" for SNMPv2.

4.2.5 Mapping to the DESCRIPTION clause

The value of the DESCRIPTION clause will be a sanitized version of the MIF Description statement. Multi-part literals will be combined into a single string according to MIF rules. Any embedded (escaped) double quotes will be replaced by a (non-escaped) single quote.

For conceptual table and row objects, the description will be formed from the MIF Description statement for the group definition.

4.2.6 Mapping to the REFERENCE clause

The value of the REFERENCE clause will be the group class string embedded in single quotes followed by a space and the attribute id.

4.2.7 Mapping to the INDEX clause

Each DMI component may have multiple instances of a particular group class, hence a conceptual table derived from a group with no keys will have an INDEX clause of the form:

```
INDEX {compId, groupId}
```

where compId and groupId are imported from the DMTF-DMI-MIB.

If the conceptual table is derived from a group class definition that specifies one or more keys (a DMTF table), then the ordered set of columnar objects corresponding to those keys form the remainder of the INDEX clause:

```
INDEX {compId, groupId, <key1>, ...,<keyN>}
```

4.2.8 Mapping RowStatus OBJECT

If the DMI class contains one or more key attributes with access equal to read-write or write-only, instances of that DMI class may be created or deleted through the management interface (e.g., using the DmiAddRow and DmiDeleteRow MI functions).

To preserve this capability in the translated SNMP interface for the DMI class, an additional columnar object is defined in the resulting SNMP conceptual table. When SNMP SetRequest operations are performed on this object, the SNMP-to-DMI Mapping Agent carries out the operation by invoking the appropriate DMI MI functions to add or delete the specified row in the table. Note, this new columnar object is defined only in the SNMP MIB translation; the DMI MIF definition is not modified in any way.

In accord with the SNMPv2 SMI standard, this additional columnar object is defined to be of type "RowStatus" and has MAX-ACCESS "read-create". If the DMI MIF is translated using SNMPv1 SMI, the ACCESS is defined as "read-write". Further, implementation of the RowStatus object by the SNMP-to-DMI Mapping Agent must comply with the rules defined in RFC 1902 [5]. See also section 5.6 of this specification.

The <descriptor> for the RowStatus object is formed by applying the name mapping algorithm defined above to the "specific-name" portion of the DMI class string; then appending the word "State".

The Object Identifier for the RowStatus object is assigned sub-identifier zero (0). Hence, the complete OID is defined as:

```
OID = {grpOID.1.0}
```

Zero is a reserved attribute id value in both SNMP and DMI, so assigning zero to the RowStatus object sub-id will not conflict with the attribute ids of current or future attributes defined in the DMI class.

4.3 Event Mapping

This mapping assumes that the MIF complies with the semantic model for DMI events defined in the DMTF document "DMI 2.0 Specification" [14].

A group class string of the form "EventGeneration|*|*" indicates that the group is not a normal MIF data group, but rather is a special Event Generation group that formally describes a related set of DMI Events. All such Event Generation groups are constructed according to the skeleton in [14], but each has a unique class string, and different sets of values for the "Event Type", "Event System", "Event Subsystem" and "Event Solution" enumerations.

Each Event Generation group class becomes a set of SNMP traps (notifications) and a set of non-columnar objects that are passed as varBinds of the traps (notifications). The OID assigned to the group class {grpOID} is used to identify the traps and varBinds as described below.

An SNMP TRAP-TYPE macro (SNMPv1) or NOTIFICATION-TYPE macro (SNMPv2) is generated for each enumerated value of the "Event Type" attribute of the group.

For SNMPv1, the OID assigned to the Event Generation group class becomes the "Enterprise ID", and the enumerated values of the "Event Type" attribute become the specific trap numbers. The SNMP <descriptor> for the Enterprise ID is formed by applying the name mapping algorithm defined above on the 'specific-name-of-assoc-group' portion of the event class string. If the grpOID value of the Event Generation group is the same as the OID for the associated group, the SNMP <descriptor> should be left the same as for the associated group.

```
ENTERPRISE grpOID
Generic trap #  = Enterprise Specific (6)
Specific trap # = Event Type
```

For SNMPv2, the OID assigned to each trap is formed by concatenating the OID of the group, a sub-id of zero, and the value of the "Event Type" attribute.

```
OID = {grpOID.0.EventType}
```

The SNMP <descriptor> of the two OIDs that prefix the specific Notification Type OIDs are formed by applying the name mapping algorithm defined above on the "specific-name-of-assoc-group" portion of the event class string. The <descriptor> associated with the grpOID is the same as that defined for the SNMPv1 Enterprise ID. The <descriptor> associated with grpOID.0 is formed by applying the name mapping algorithm above on the "specific-name-of-assoc-group" portion of the event class string, then appending the text "Traps".

For the other enumerated attributes in the MIF event group: "Event System", "Event Subsystem", and "Event Solution" (if present); a group of (non-columnar) SNMP objects is generated using the OID of the group class as the prefix, and the id of each attribute as the final sub-id:

```
OID = {grpOID.attrId}
```

The SNMP <descriptor> for these objects is formed by applying the name mapping algorithm defined above on the 'specific-name-of-assoc-group' portion of the class string, then appending the string "EvSys", "EvSub" or "EvSol", respectively.

These objects have an access type of "not-accessible" for SNMPv1 and "accessible-for-notify" for SNMPv2. The other clauses are generated as described in "Managed Objects Mapping".

4.3.1 Mapping to the TRAP-TYPE or NOTIFICATION-TYPE <descriptor>

The descriptor is formed by applying the name mapping algorithm defined above on the 'specific-name-of-assoc-group' portion of the event class string and on the appropriate enumeration literal (string) of the "Event Type" attribute, then appending the latter to the former.

If the resulting name is longer than the 64-character maximum for SNMP names, the descriptor is instead formed by applying the name mapping algorithm on the ‘specific-name-of-assoc-group’ portion of the event class string, then appending the text "Ev" followed by the numeric value of the event type.

4.3.2 Mapping to the VARIABLES/OBJECTS clause

The VARIABLES clause for SNMPv1 and the OBJECTS clause for SNMPv2 contain the following ordered set of varBinds: dmiEventDateTime, dmiCompId, dmiEventSeverity, dmiEventStateIndex, dmiEventAssocGroup, eventSystem, and eventSubsystem.

When the trap (notification) is generated, the mapping agent may also append any of the following varbinds if the corresponding attribute is present in the DMI Event: eventSolution, dmiEventVendorMsg, dmiEventVendorData.

If the DMI event contains event-specific attributes such as a key list or additional information, those attributes are translated to SNMP according to the rules for OBJECT-TYPE definitions and appended to the SNMP notification as well.

Note that the var-bind objects who's name starts with "dmiEvent..." are defined in the DMTF-DMI-MIB and are shared by all traps.

4.3.3 Mapping to the STATUS clause

The value of the STATUS clause will always be "mandatory" for SNMPv1, and "current" for SNMPv2.

4.3.4 Mapping to the DESCRIPTION clause

The value of the DESCRIPTION clause will be the enumeration value (string) of the "Event Type" attribute.

4.3.5 Mapping to the REFERENCE clause

The value of the REFERENCE clause will be the group class string embedded in single quotes followed by a space and the numeric value of "Event Type".

4.4 Enumeration Textual Convention Mapping

DMI Enumerations defined outside the scope of an attribute definition are translated into SNMP MIBs according to the SNMP SMI version in use. In both cases, the enumeration type descriptor and enumerated value descriptors are formed by performing the name mapping defined above to the DMI enumeration Name statement and literal strings, respectively.

For SNMPv1 SMI, the enumeration is defined as an ASN.1 alias of the form:

```
      EnumName ::= INTEGER { literal(value), ... }
```

For SNMPv2 SMI, the enumeration is defined as an SNMP Textual Convention of the form:

```
      EnumName ::= TEXTUAL-CONVENTION
            STATUS            current
            DESCRIPTION       "DMI MIF Enumeration"
            REFERENCE         ""
            SYNTAX            INTEGER { literal(value}, ... }
```

Since the DMI ENUM definition does not include a description statement, the DESCRIPTION clause for the SNMPv2 textual convention shall be "DMI MIF Enumeration". The STATUS clause is "current". The REFERENCE clause is omitted, for lack of an automated translation. If the MIF is translated manually, the DESCRIPTION clause should describe what the enumeration is for and the REFERENCE clause should identify the MIF and organization that created it; e.g., "DMTF: PC System MIF".

4.4 Example

The example on the next three pages shows a hypothetical MIF - the Kennel MIF, and the SNMP MIBs produced from it by applying the mappings defined in this document. Note that two separate MIB modules are generated, one for each OID prefix.

Note that both the MIF and MIBs are represented in an abbreviated syntax for brevity.

4.4.1 Example DMI MIF

```
start component   "Pink Poodle Dog Kennel System"

start enum
      "dog type"  1="Poodle" 2="Beagle"
end enum

start group
1 "ComponentID"               class="DMTF|ComponentID|001"
   pragma                     "SNMP: 1.3.6.1.4.1.412.2.1.1;"
      1 "Manufacturer"        r/o   string(64)
            .                 .           .           .
            .                 .           .           .
      6 "Verify"              r/o   0="an error..." 1=...
end group

start group
2 "Kennel Data"               class="DMTF|KennelID|001"
   pragma                     "SNMP:1.3.6.1.4.1.412.2.999.1;"
      1 "Kennel Address"      r/o   string(128)
      2 "Number of Dogs"      r/o   integer
end group

start group
3 "Dog Data"                  class="DMTF|DogID|001"  key=1
  pragma                      "SNMP:1.3.6.1.4.1.412.2.999.2;"
      1 "Dog-tag"             r/o   integer
      2 "Dog Breed            r/o   "dog type"
      3 "Dog Name"            r/o   string(64)
      4 "Dog Sex"             r/o   1="Male" 2="Female"
end group

start group
4 "Alerts"        class="EventGeneration|DMTF^^KennelID|001"      key=4
  pragma                      "SNMP:1.3.6.1.4.1.412.2.999.3;"
      1 "Event Type"          r/o   1="Escape" 2="Sickness" 3="Death"
      2 "Event Severity"      r/o   1="Info" 2="OK" 3="Warn"...
      3 "Event is State Based"  r/o  BOOL
      4 "Event State Key"     r/o   integer
      5 "Associated Group"r/o string
      6 "Event System"  r/o   1="Staff" 2="Dog"
      7 "Event Subsystem"     r/o   1="Male" 2=Female"

end group

end component
```

4.4.2 Resulting SNMP MIB

EDITOR NOTE: Reformat to use actual SNMPv2 SMI.

```
Kennel-MIB DEFINITIONS ::= BEGIN

IMPORTS dmiCompId, dmiGrpId, DmiDate
        dmiTimeStamp, dmiEventSeverity, dmiEventStateIndex
FROM DMTF-DMI-MIB

kennel ::= OBJECT IDENTIFIER { 1 3 6 1 4 1 412 2 999 }

DogType ::= INTEGER { poodle(1), beagle(2) }

kennel.1 kennelIDTable: SEQUENCE OF KennelIDEntry
kennel.1.1 kennelIDEntry: SEQUENCE KennelIDEntry
           INDEX { dmiCompId, dmiGrpId }
kennel.1.1.1 kennelAddress: InternationalDisplayString(SIZE(0..128)
kennel.1.1.2 numberOfDogs: INTEGER

kennel.2 dogIDTable: SEQUENCE OF DogIDEntry
kennel.2.1 dogIDEntry: SEQUENCE DogIDEntry
           INDEX { dmiCompId, dmiGrpId, dogTag }
kennel.2.1.1 dogTag: INTEGER
kennel.2.1.2 dogBreed: DogType
kennel.2.1.3 dogName: InternationalDisplayString(0..64)
kennel.2.1.4 dogSex: INTEGER { male(1), female(2)}

 -- trap related stuff
kennel.3.5 kennelIDEventSystem: INTEGER { staff(1), dog(2) }
kennel.3.6 kennelIDEventSubsystem: INTEGER { male(1), female(2) }

enterprise: KennelID kennel.3
  trap: kennelIDEscape, value 1
  trap: kennelIDSickness, value 2
  trap: kennelIDDeath, value 3

END
```

4.4.3 SNMP MIB for DMI Component ID Group

```
Component-MIB DEFINITIONS ::= BEGIN

IMPORTS dmiCompId, dmiGrpId, DmiDate FROM DMTF-DMI-MIB

component ::= OBJECT IDENTIFIER { 1 3 6 1 4 1 412 2 1 }

component.1 componentIDTable SEQUENCE OF ComponentIDEntry
component.1.1 componentIDEntry SEQUENCE ComponentIDEntry
              INDEX { dmiCompId, dmiGrpId }
component.1.1.1 manufacturer: IntlDisplayString(SIZE(0..64))
component.1.1.2 product: IntlDisplayString(SIZE(0..64))
component.1.1.3 version: IntlDisplayString(SIZE(0..64))
component.1.1.4 serialNumber: IntlDisplayString(SIZE(0..64))
component.1.1.5 installation: DmiDate
component.1.1.6 verify: INTEGER { anErrorOccurredCheckStatusCode(0),
                                  componentDoesNotExist(1),
                                          ... }
END
```

5 Mapping Agent Operation

The SNMP-to-DMI Mapping Agent is a piece of code that resides on the managed node along with the DMI instrumentation, the Service Provider, and the SNMP master agent. The mapping agent appears as an SNMP sub-agent to the SNMP master agent, and as a management application to the DMI Service Provider, and acts as a broker between them. Alternatively, the mapping agent can reside on another system acting as an SNMP-to-DMI proxy for the managed system, provided the managed system supports one of the DMI 2.0 remote access RPCs.

The mapping agent implements the DMTF-DMI-MIB that is defined in this document and also provides SNMP access to any DMI data on the managed node. The mapping agent may provide access to the DMI data either through customize implementation for each DMI MIF Group, or through a general mapping procedure. Either way, the mapping agent is able to perform these functions using only information available through the MI.

The mapping agent registers with the DMI Service Provider as a management application and uses the MI calls to discover all registered group classes and their associated OIDs. It then registers with the SNMP master agent for all OID sub-trees supported by itself (DMTF-DMI-MIB) and the DMI instrumentation.

When the mapping agent receives an SNMP request from the master agent for one of the objects it handles, it will service the request by making the appropriate calls to the MI. When it receives an indication (event) from the Service Provider, it will translate it into the appropriate SNMP trap (notification) and send it to the master agent. A key requirement of the mapping agent is that it must preserve the semantics of all SNMP operations defined by the SNMP version implemented by the mapping agent.

5.1 Object Identifier Mapping

To function, the mapping agent needs to maintain a mapping between OIDs and all registered group class strings. This mapping is obtained using a combination of three mechanisms:

1. Built in table for some standard MIFs.

2. The SNMP pragma specified in the group definition.

3. Generating unique OIDs for any registered group class that is not assigned an OID though the other two methods.

5.1.1 Built-in Table of OIDs

All mapping agent implementations shall contain class string to OID mappings for all DMTF standard groups defined at the time this specification was issued, as listed in the following tables. A mapping agent may also implement a local configuration store where these OID mappings may be augmented at run-time. These measures allow the mapping agent to operate in existing DMI implementations where those MIFs already installed in the DMI Service Provider do not contain embedded pragmas.

Note, the following mapping definitions are identical to the pragma assignments specified in the MASTER.MIF revised October 1997.

Desktop Management Interface Specification [14]

Group Class String	OBJECT IDENTIFIER
"DMTF|ComponentID|"	1.3.6.1.4.1.412.2.1.1
"DMTF|Event State|"	1.3.6.1.4.1.412.2.1.2
"DMTF|SP Indication Subscription|"	1.3.6.1.4.1.412.2.1.3
"DMTF|SPFilterInformation|"	1.3.6.1.4.1.412.2.1.4

LAN Adapter Standard MIF [15]

Group Class String	OBJECT IDENTIFIER
"DMTF|Network Adapter 802 Port|"	1.3.6.1.4.1.412.2.2.1
"DMTF|802 Alternate Address|"	1.3.6.1.4.1.412.2.2.2
"DMTF|Network Adapter Driver|"	1.3.6.1.4.1.412.2.2.3
"DMTF|Network Adapter Hardware|"	1.3.6.1.4.1.412.2.2.4
"DMTF|Boot ROM Configuration|"	1.3.6.1.4.1.412.2.2.5
"DMTF|Boot ROM Capabilities|"	1.3.6.1.4.1.412.2.2.6

Software Standard MIF [16]

Group Class String	OBJECT IDENTIFIER
"DMTF|Software Component Information|"	1.3.6.1.4.1.412.2.3.1
"DMTF|Software Signature|"	1.3.6.1.4.1.412.2.3.2
"DMTF|Location|"	1.3.6.1.4.1.412.2.3.3
"DMTF|Equivalence|"	1.3.6.1.4.1.412.2.3.4
"DMTF|Superseded components|"	1.3.6.1.4.1.412.2.3.5
"DMTF|Maintenance|"	1.3.6.1.4.1.412.2.3.6
"DMTF|Verification|"	1.3.6.1.4.1.412.2.3.7
"DMTF|Subcomponents|"	1.3.6.1.4.1.412.2.3.8
"DMTF|Component Dependencies|"	1.3.6.1.4.1.412.2.3.9
"DMTF|Attribute Dependencies|"	1.3.6.1.4.1.412.2.3.10
"DMTF|File List|"	1.3.6.1.4.1.412.2.3.11
"DMTF|Installation|"	1.3.6.1.4.1.412.2.3.12
"DMTF|Installation Log Files|"	1.3.6.1.4.1.412.2.3.13
"DMTF|Support|"	1.3.6.1.4.1.412.2.3.14

Systems Standard MIF [17,18]

Group Class String	OBJECT IDENTIFIER
"DMTF|General Information|"	1.3.6.1.4.1.412.2.4.1
"DMTF|Operating System|"	1.3.6.1.4.1.412.2.4.2
"DMTF|System BIOS|"	1.3.6.1.4.1.412.2.4.3
"DMTF|BIOS Characteristic|"	1.3.6.1.4.1.412.2.4.4
"DMTF|Processor|"	1.3.6.1.4.1.412.2.4.5
"EventGeneration|DMTF^^Processor|"	1.3.6.1.4.1.412.2.4.5
"DMTF|Motherboard|"	1.3.6.1.4.1.412.2.4.6
"EventGeneration|DMTF^^Motherboard|"	1.3.6.1.4.1.412.2.4.6

"DMTF|Physical Memory|"	1.3.6.1.4.1.412.2.4.7
"DMTF|Logical Memory|"	1.3.6.1.4.1.412.2.4.8
"EventGeneration|DMTF^^Logical Memory|"	1.3.6.1.4.1.412.2.4.8
"DMTF|System Cache|"	1.3.6.1.4.1.412.2.4.9
"EventGeneration|DMTF^^System Cache|"	1.3.6.1.4.1.412.2.4.9
"DMTF|Parallel Ports|"	1.3.6.1.4.1.412.2.4.10
"DMTF|Serial Ports|"	1.3.6.1.4.1.412.2.4.11
"DMTF|IRQ|"	1.3.6.1.4.1.412.2.4.12
"DMTF|DMA|"	1.3.6.1.4.1.412.2.4.13
"DMTF|Memory Mapped I/O|"	1.3.6.1.4.1.412.2.4.14
"DMTF|System Enclosure|"	1.3.6.1.4.1.412.2.4.15
"DMTF|Power Supply|"	1.3.6.1.4.1.412.2.4.16
"EventGeneration|DMTF^^Power Supply|"	1.3.6.1.4.1.412.2.4.16
"DMTF|Cooling Device|"	1.3.6.1.4.1.412.2.4.17
"EventGeneration|DMTF^^Cooling Device|"	1.3.6.1.4.1.412.2.4.17
"DMTF|System Slot|"	1.3.6.1.4.1.412.2.4.18
"DMTF|System Slots|"	1.3.6.1.4.1.412.2.4.18
"DMTF|Video|"	1.3.6.1.4.1.412.2.4.19
"DMTF|Video BIOS|"	1.3.6.1.4.1.412.2.4.20
"DMTF|Video BIOS Characteristic|"	1.3.6.1.4.1.412.2.4.21
"DMTF|Disks|"	1.3.6.1.4.1.412.2.4.22
"EventGeneration|DMTF^^Disks|"	1.3.6.1.4.1.412.2.4.22
"DMTF|Disks Mapping Table|"	1.3.6.1.4.1.412.2.4.23
"DMTF|Partition|"	1.3.6.1.4.1.412.2.4.24
"DMTF|Disk Controller|"	1.3.6.1.4.1.412.2.4.25
"EventGeneration|DMTF^^Disk Controller|"	1.3.6.1.4.1.412.2.4.25
"DMTF|Logical Drives|"	1.3.6.1.4.1.412.2.4.26
"DMTF|Mouse|"	1.3.6.1.4.1.412.2.4.27
"DMTF|Keyboard|"	1.3.6.1.4.1.412.2.4.28
"DMTF|FRU|"	1.3.6.1.4.1.412.2.4.29
"DMTF|Operational State|"	1.3.6.1.4.1.412.2.4.30
"DMTF|System Resources Description|"	1.3.6.1.4.1.412.2.4.31
"DMTF|System Resources|"	1.3.6.1.4.1.412.2.4.32
"DMTF|Physical Memory Array|"	1.3.6.1.4.1.412.2.4.33
"EventGeneration|DMTF^^Physical Memory Array|"	1.3.6.1.4.1.412.2.4.33
"DMTF|Memory Array Mapped Addresses|"	1.3.6.1.4.1.412.2.4.34
"DMTF|Memory Device|"	1.3.6.1.4.1.412.2.4.35
"DMTF|Memory Device Mapped Addresses|"	1.3.6.1.4.1.412.2.4.36
"DMTF|System Resources 2|"	1.3.6.1.4.1.412.2.4.37
"DMTF|System Resource Device Info|"	1.3.6.1.4.1.412.2.4.38
"DMTF|System Resource Memory Info|"	1.3.6.1.4.1.412.2.4.39
"DMTF|System Resource I/O Info|"	1.3.6.1.4.1.412.2.4.40
"DMTF|System Resource IRQ Info|"	1.3.6.1.4.1.412.2.4.41
"DMTF|System Resource DMA Info|"	1.3.6.1.4.1.412.2.4.42
"DMTF|Mass Store Mapping Table|"	1.3.6.1.4.1.412.2.4.43
"DMTF|Mass Store Segment Table|"	1.3.6.1.4.1.412.2.4.44
"DMTF|Mass Store Logical Drives Table|"	1.3.6.1.4.1.412.2.4.45
"EventGeneration|DMTF^^Mass Store Logical Drives Table|"	1.3.6.1.4.1.412.2.4.45
"DMTF|Mass Store Array Info Table|"	1.3.6.1.4.1.412.2.4.46
"DMTF|Sequential Access Devices|"	1.3.6.1.4.1.412.2.4.47
"DMTF|System Reset|"	1.3.6.1.4.1.412.2.4.48
"EventGeneration|DMTF^^System Reset|"	1.3.6.1.4.1.412.2.4.48

"DMTF|Sysetm Hardware Security|"	1.3.6.1.4.1.412.2.4.49
"EventGeneration|DMTF^^System Hardware Security|"	1.3.6.1.4.1.412.2.4.49
"DMTF|Power Supply Output Voltage Capabilities|"	1.3.6.1.4.1.412.2.4.50
"DMTF|System Power Controls|"	1.3.6.1.4.1.412.2.4.51
"DMTF|UPS Battery|"	1.3.6.1.4.1.412.2.4.52
"EventGeneration|DMTF^^UPS Battery|"	1.3.6.1.4.1.412.2.4.52
"DMTF|Voltage Probe|"	1.3.6.1.4.1.412.2.4.53
"EventGeneration|DMTF^^Voltage Probe|"	1.3.6.1.4.1.412.2.4.53
"DMTF|Temperature Probe|"	1.3.6.1.4.1.412.2.4.54
"EventGeneration|DMTF^^Temperature Probe|"	1.3.6.1.4.1.412.2.4.54
"DMTF|Electrical Current Probe|"	1.3.6.1.4.1.412.2.4.55
"EventGeneration|DMTF^^Electrical Current Probe|"	1.3.6.1.4.1.412.2.4.55
"DMTF|Diagnostic Function Group|"	1.3.6.1.4.1.412.2.4.56
"DMTF|Diagnostic Request Group|"	1.3.6.1.4.1.412.2.4.57
"DMTF|Diagnostic Results Group|"	1.3.6.1.4.1.412.2.4.58
"DMTF|Diagnostic Results FRU Call|"	1.3.6.1.4.1.412.2.4.59
"DMTF|Out-of-band Remote Access|"	1.3.6.1.4.1.412.2.4.60
"DMTF|Cache Performance Table|"	1.3.6.1.4.1.412.2.4.61
"DMTF|Bus Utilization Table|"	1.3.6.1.4.1.412.2.4.62
"DMTF|Physical Container Global Table|"	1.3.6.1.4.1.412.2.4.63
"EventGeneration|DMTF^^Physical Container Global Table|"	1.3.6.1.4.1.412.2.4.63
"DMTF|Bus Global Table|"	1.3.6.1.4.1.412.2.4.64
"DMTF|Physical Expansion Site Table|"	1.3.6.1.4.1.412.2.4.65
"DMTF|Power Unit Global Table|"	1.3.6.1.4.1.412.2.4.66
"DMTF|Cooling Unit Global Table|"	1.3.6.1.4.1.412.2.4.67
"DMTF|Structure Dependency Table|"	1.3.6.1.4.1.412.2.4.68
"EventGeneration|DMTF^^Structure Dependency Table|"	1.3.6.1.4.1.412.2.4.68
"DMTF|Bus Card Location|"	1.3.6.1.4.1.412.2.4.69
"DMTF|System Contact Information|"	1.3.6.1.4.1.412.2.4.70
"DMTF|System Memory Settings|	1.3.6.1.4.1.412.2.4.71

Large Mailing Operations [19]

Group Class String ----------	OBJECT IDENTIFIER ----------
"DMTF|General|"	1.3.6.1.4.1.412.2.5.1
"DMTF|Localization Table|"	1.3.6.1.4.1.412.2.5.2
"DMTF|Finisher Device Table|"	1.3.6.1.4.1.412.2.5.3
"DMTF|Finisher Input Table|"	1.3.6.1.4.1.412.2.5.4
"DMTF|Finishing Features|"	1.3.6.1.4.1.412.2.5.5
"DMTF|Finisher Output Table|"	1.3.6.1.4.1.412.2.5.6
"DMTF|Finisher Supply Table|"	1.3.6.1.4.1.412.2.5.7
"DMTF|Finisher Colorant Table|"	1.3.6.1.4.1.412.2.5.8
"DMTF|Alert Table|"	1.3.6.1.4.1.412.2.5.9
"DMTF|Insert Feeder Table|"	1.3.6.1.4.1.412.2.5.10
"DMTF|Envelope Feeder Table|"	1.3.6.1.4.1.412.2.5.11
"DMTF|Output Device Table|"	1.3.6.1.4.1.412.2.5.12

"DMTF|Output Device Supply Table|"	1.3.6.1.4.1.412.2.5.13
"DMTF|Output Device Colorant Table|"	1.3.6.1.4.1.412.2.5.14
"DMTF|ProductTable|"	1.3.6.1.4.1.412.2.5.15
"DMTF|Process Path Table|"	1.3.6.1.4.1.412.2.5.16
"DMTF|Process History Table|"	1.3.6.1.4.1.412.2.5.17
"DMTF|Print History Table|"	1.3.6.1.4.1.412.2.5.18
"DMTF|Insert History Table|"	1.3.6.1.4.1.412.2.5.19
"DMTF|Print Job Table|"	1.3.6.1.4.1.412.2.5.20
"DMTF|Insert Job Table|"	1.3.6.1.4.1.412.2.5.21
"DMTF|Bill of Materials Table|"	1.3.6.1.4.1.412.2.5.22
"DMTF|Mailpiece XReference Table|"	1.3.6.1.4.1.412.2.5.23

Monitor Standard MIF [20]

Group Class String	OBJECT IDENTIFIER
"DMTF|Monitor Additional Informations|"	1.3.6.1.4.1.412.2.6.1
"DMTF|Monitor Resolutions|"	1.3.6.1.4.1.412.2.6.2

Printer Standard MIF [21]

Group Class String	OBJECT IDENTIFIER
"DMTF|Printer Interface Table|"	1.3.6.1.4.1.412.2.7.1
"DMTF|Storage Table|"	1.3.6.1.4.1.412.2.7.2
"DMTF|Device Table|"	1.3.6.1.4.1.412.2.7.3
"DMTF|Printer General|"	1.3.6.1.4.1.412.2.7.4
"DMTF|Cover Table|"	1.3.6.1.4.1.412.2.7.5
"DMTF|Printer Localization Table|"	1.3.6.1.4.1.412.2.7.6
"DMTF|Input Table|"	1.3.6.1.4.1.412.2.7.7
"DMTF|Output Table|"	1.3.6.1.4.1.412.2.7.8
"DMTF|Marker Table|"	1.3.6.1.4.1.412.2.7.9
"DMTF|Marker Supplies Table|"	1.3.6.1.4.1.412.2.7.10
"DMTF|Marker Colorant Table|"	1.3.6.1.4.1.412.2.7.11
"DMTF|Media Path Table|"	1.3.6.1.4.1.412.2.7.12
"DMTF|Channel Table|"	1.3.6.1.4.1.412.2.7.13
"DMTF|Interpreter Table|"	1.3.6.1.4.1.412.2.7.14
"DMTF|Display Buffer Table|"	1.3.6.1.4.1.412.2.7.15
"DMTF|Console Lights Table|"	1.3.6.1.4.1.412.2.7.16
"DMTF|Printer Alert Table|"	1.3.6.1.4.1.412.2.7.17

Mobile Computer Supplement MIF [22]

Group Class String	OBJECT IDENTIFIER
"DMTF|Portable Battery|"	1.3.6.1.4.1.412.2.8.1
"EventGeneration|DMTF^^Portable Battery|"	1.3.6.1.4.1.412.2.8.1
"DMTF|Dynamic States|"	1.3.6.1.4.1.412.2.8.2
"DMTF|Video Output Device|"	1.3.6.1.4.1.412.2.8.3
"DMTF|Infrared Port|"	1.3.6.1.4.1.412.2.8.4
"DMTF|Pointing Device|"	1.3.6.1.4.1.412.2.8.5

"DMTF|System Power Management|"	1.3.6.1.4.1.412.2.8.6
"DMTF|Power Management Table|"	1.3.6.1.4.1.412.2.8.7
"DMTF|Power Management Binary Association|"	1.3.6.1.4.1.412.2.8.8
"DMTF|Device Bay|"	1.3.6.1.4.1.412.2.8.9

5.1.2 OIDs Acquired through MI

The mapping agent will iterate through all registered group classes in the MIF database and extract the mapping information (if any) from the SNMP pragma statement.

5.1.3 Dynamically Generated OIDs

For each group class that does not have an administratively assigned SNMP OID mapping, the procedure defined in section 3.2 is used to dynamically generate a unique OID for the group class.

5.2 Instance Identifier Mapping

SNMP identifies a specific instance of an attribute by appending an OID fragment, called the instance identifier, to the OID of the attribute definition. The instance identifier is constructed by first applying the SYNTAX and INDEX mapping rules defined in sections 4.2.2 and 4.2.7, then by applying the rules specified in Section 7.7 of RFC 1902 [5] to produce the object identifier representation of the index values.

For convenience, construction of the SNMP instance identifier for a DMI Attribute represented as an SNMP variable binding is summarized as follows:

1. The first sub-identifier is assigned the value of the DMI Component Id.
2. The second sub-identifier is assigned the value of the DMI Group Id.
3. For each of the DMI Key attributes listed in the INDEX mapping, one or more sub-identifiers are appended according to the following conversion rules:
   - DMI Integer-typed keys are represented by a single sub-identifier with value equal to that of the DMI attribute.
   - DMI Integer64-typed keys are mapped into SNMP as a fixed-length sequence of eight octets. Therefore, an Integer64 value is represented by exactly eight sub-identifiers, each sub-identifier equal to the respective octet value.
   - DMI Date-typed keys are mapped into SNMP as a fixed-length sequence of 25 octets. Therefore, a DmiDate value is represented by exactly 25 sub-identifiers, each sub-identifier equal to the respective octet value.
   - DMI Octet String- and DMI Display String-typed keys are mapped into SNMP as a variable-length sequence of octets. Therefore, a string is represented by ‘n+1’ sub-identifiers, the first sub-identifier equal to the length of the octet string, and the remaining n sub-identifiers equal to the respective octet value in the octet string.

5.3 GetRequest Mapping

The mapping agent must preserve the semantics of the SNMP GetRequest-PDU. For the most part, this is a straight-forward translation between SNMP and DMI. However, attention must be given to SNMP’s requirement that the Response-PDU include the variable bindings in the same order as the original request, and so on.

If a DMI error occurs while servicing the SNMP request, an appropriate SNMP error value/code is returned in the SNMP Response-PDU. Error code mappings for the GetRequest-PDU are shown in the following table. Any other DMI error should probably result in a genErr. System specific errors should be mapped appropriately.

DMI Error	SNMPv1	SNMPv2
0x00001 More Data	tooBig	tooBig *1
0x00100 Attr not found	noSuchName	noSuchInstance *2
0x00101 Val exceeds max	genErr	genErr
0x00102 Comp instr not found	noSuchName	noSuchInstance *2
0x00103 Enum error	genErr	genErr
0x00104 Group not found	noSuchName	noSuchInstance *2
0x00105 Illegal keys	noSuchName	noSuchInstance
0x00106 Illegal to set	genErr	genErr
0x00107 Can't resolve attr fn	noSuchName	noSuchInstance *3
0x00108 Illegal to get	noSuchName	noSuchObject
0x00109 No description	genErr	genErr
0x0010A Row not found	noSuchName	noSuchInstance
0x0010B Direct I/f not reg'd	noSuchName	noSuchInstance *3
0x0010C MIF DB corrupt	genErr	genErr
0x0010D Attr not supported	noSuchName	noSuchInstance
0x0020B Insufficient Priv	noSuchName	noSuchInstance *3
0x0020C No access fn	noSuchName	noSuchInstance *3
0x01001 Overlay not found	noSuchName	noSuchInstance *3
0x10002 Overlay not found	noSuchName	noSuchInstance *3
0x10004 component get err	noSuchName	noSuchInstance *3
0x10005 component key err	noSuchName	noSuchInstance
0x10008 component row err	noSuchName	noSuchInstance

Notes:
1. Assumes DMI confirm buffer is > max PDU size
2. Could be noSuchObject if no components support it, but its kind of hard to tell
3. Component instrumentation not accessible, could be noSuchObject if no other component supports it.

5.4 GetNext and GetBulk Request Mapping

The mapping agent must preserve the semantics of the SNMP GetNextRequest-PDU and GetBulkRequestPDU. Special attention must be given to SNMP's requirement for lexicographic ordering, because the DMI does not define any ordering for the way keyed rows are returned when iterating through a table with DmiGetNextRow, nor does it allow partial keys. Fortunately, the DmiList commands for components, groups, and attributes are compatible with the requirements of SNMP.

If a DMI error occurs while servicing the SNMP request, an appropriate SNMP error value/code is returned in the SNMP Response-PDU. Error code mappings for the GetNetxtRequest-PDU and GetBulkRequest-PDU are the same as for the GetRequest-PDU (see above), except that instead of returning noSuchObject, noSuchInstance or noSuchName the lexicographically next object should be returned. System specific errors should be mapped appropriately.

5.5 Set Request Mapping

The mapping agent must preserve the semantics of the SNMP SetRequest-PDU by making appropriate use of the Set, Release and Reserve DmiSetModes with the DmiSetAttribute operation. If the DmiSetAttribute operation fails for any attribute, the mapping agent must attempt to rollback the values of any attributes already set as part of that SetRequest-PDU.

If a DMI error occurs while servicing the SNMP request, an appropriate SNMP error value/code is returned in the SNMP Response-PDU. Mappings for errors codes returned from a DmiSetReserveAttributeCmd are shown in the following table; any other DMI error should result in a genErr. If an error is returned from the DmiSetAttributeCmd, then either commitFailed or undoFailed should be returned as appropriate.

DMI Error	SNMPv1	SNMPv2
0x00100 Attr not found	noSuchName	noAccess
0x00101 Val exceeds max		wrongLength
0x00102 Comp instr not found		noAccess
0x00103 Enum error		wrongValue
0x00104 Group not found		noAccess
0x00105 Illegal keys		noAccess
0x00106 Illegal to set		notWritable *1
0x00107 Can't resolve attr fn		noAccess
0x00108 Illegal to get		noSuchObject
0x0010A Row not found		noAccess
0x0010B Direct I/f not reg'd		resourceUnavailable
0x0010C MIF DB corrupt		genErr
0x0010D Attr not supported		noAccess
0x0020B Insufficient Priv		noAccess
0x0020C No access fn		resourceUnavailable
0x01001 Overlay not found		resourceUnavailable
0x10002 Overlay not found		resourceUnavailable
0x10005 component key err		noAccess
0x10008 component row err		noAccess
0x10009 component set err		notWritable *1

Notes:
1. Use DmiListAttributeCmd to determine exact cause of error and return notWritable, wrongType, wrongLength, or wrongValue as appropriate.

5.6 Row Addition/Deletion Mapping

Section 4.2.8 describes the addition of an SNMP RowStatus columnar object to the resulting SNMP conceptual table for DMI classes that permit row addition and deletion through the management interface.

When the mapping agent receives a SetRequest operation on the RowStatus object of a conceptual table, the mapping agent must translate the request into appropriate uses of the DMI MI DmiAddRow and DmiDeleteRow operations in accord with the semantics defined by RFC 1902 [5] for the RowStatus textual convention. The DMI Set, Reserve

and Release modes of the DmiSetAttribute operation should also be used when changing the RowStatus state for row modification.

5.7 Notification Mapping

The mapping agent converts DMI Events and Indications into SNMP Traps according to the following elements of procedure. If the mapping agent supports SNMPv2, construct an SNMPv2 Trap; otherwise construct an equivalent SNMPv1 Trap by applying the SNMPv2 to SNMPv1 rules specified in RFC 1908 [11].

5.7.1 Mapping DMI Events

Upon receipt of a DMI Event, an attempt is made to resolve the Event Generation group class name into an assigned SNMP OID either from the set of standard MIF assignments or the SNMP pragma statement in the MIF definition. If an OID mapping exists, the DMI Event is translated into an event-specific SNMP notification as described below. Otherwise, an instance of the general-purpose dmiEventIndication notification is created from the DMI Event.

To map the DMI Event into an event-specific SNMP notification, construct an SNMPv2 Trap PDU with the following variable bindings (varbinds). The first two varbinds are mandatory, as defined by the SNMPv2 standard [8], and serve to identify an instance of the event. The data varbinds follow. The first seven data varbinds are always present in DMI Events. They convey mandatory information from the DMI Event Generation class. The remaining varbinds are conditionally included based on the presence of the corresponding information in the DMI Event.

SNMPv2 Standard Varbinds:

1. The first standard varbind is an instance of sysUpTime. The value is the value of the SNMP master agent's sysUpTime at the time the trap is generated.
2. The second standard varbind is an instance of snmpTrapOID. The value is an object identifier equal to <grpOID>.0.<eventType>; where, grpOID is the object identifier associated with the DMI Event class name, and eventType is the value of the DMI Event Type attribute.

Mandatory DMI Event Generation Attribute Varbinds:

1. The first data varbind is an instance of dmiEventDateTime. The value is a DMI Timestamp assigned from the timestamp parameter of the DMI Event indication.
2. The second data varbind is an instance of dmiCompId. The value is the Component Id associated with the received DMI Event.
3. The third data varbind is an instance of dmiEventSeverity. The value is the integer (enumeration) assigned from the DMI Event Severity attribute.
4. The fourth data varbind is an instance of dmiEventStateIndex. The value is -1 if the DMI Event Is State Based attribute is FALSE; otherwise, the value is assigned from the DMI Event State Key attribute.
5. The fifth data varbind is an instance of dmiEventAssociatedGroup. The value is assigned the class string from the DMI Event Associated Group attribute.
6. The sixth data varbind is an instance of the DMI event-specific Event System attribute.
7. The seventh data varbind is an instance of the DMI event-specific Event Subsystem attribute.

Optional DMI Event Generation Attribute Varbinds:

Zero or more of the following varbinds may appear after the mandatory varbinds described above if the corresponding attribute is present in the DMI Event. They are:

1. The first optional varbind is an instance of the DMI event-specific Event Solution attribute. The value is assigned the integer enumeration from the DMI Event Solution attribute.
2. The second optional varbind is an instance of dmiEventVendorMsg. The value is assigned the display string value from the DMI Event Vendor Specific Message attribute.
3. The third optional varbind is an instance of dmiEventVendorData. The value is assigned the octet string value from the DMI Event Vendor Specific Data attribute.

NOTE: all of the event varbinds described thus far, except dmiCompId, are scalars; therefore, the identifying OID for each varbind includes the ".0" instance specifier. Since dmiCompId is the index object for the dmiComponentTable, its instance specifier is the value of the component identifier.

Event Instance-specific Varbinds:

If the DMI Event includes a second DmiRowData structure containing instance-specific row data, each DMI attribute in that row data is translated into an SNMP varbind instance according to the rules specified in section 4.2 for mapping object instances from DMI to SNMP. In particular, the instance indentifier portion of varbind OID must conform to the conventions specified in section 4.2.7.

Event Source Varbinds:

The DMI 2.0 standard makes provision for an Event to be forwarded from its original source to its eventual destination by including a third DmiRowData structure in the DMI Event that identifies the original event source. In such cases, the resulting SNMP notification must include identification of the event source to distinguish the source from the system where the mapping agent is running. A similar need also exists in cases where the mapping agent is running on a system other than the managed system, even though the DMI Event is received directly from the managed system. Since the two cases are indistinguishable to recipients of the resulting SNMP notification, the following approach is always used instead of the proxy conventions described in section 1.3.

If the DMI Event originates from a system other than the system where the mapping agent resides, one last varbind, dmiEventSource, is appended to the SNMP notification to identify the original source of the DMI Event. Otherwise, this varbind is not included in the SNMP notification; and the DMI Event source is identified by the source address of the SNMP Notification itself.

5.7.2 Mapping DMI Component Added Indication

Upon receipt of a DMI Component Added indication, construct an SNMPv2 Trap as specified by the dmiComponentAddedIndication NOTIFICATION-TYPE definition. The associated data varbinds are assigned from the indication parameters as follows:

1. The first varbind is an instance of dmiCompId.
2. The second varbind is an instance of dmiCompName.

5.7.3 Mapping DMI Component Deleted Indication

Upon receipt of a DMI Component Deleted indication, construct an SNMPv2 Trap as specified by the dmiComponentDeletedIndication NOTIFICATION-TYPE definition. The associated data varbinds are from the indication parameters as follows:

1. The only varbind is an instance of dmiCompId.

5.7.4 Mapping DMI Group Added Indication

Upon receipt of a DMI Group Added indication, construct an SNMPv2 Trap as specified by the dmiGroupAddedIndication NOTIFICATION-TYPE definition. The associated data varbinds are assigned from the indication parameters as follows:

1. The first varbind is an instance of dmiCompId.
2. The second varbind is an instance of dmiGroupId
3. The second varbind is an instance of dmiGroupClassString.

5.7.5 Mapping DMI Group Deleted Indication

Upon receipt of a DMI Group Deleted indication, construct an SNMPv2 Trap as specified by the dmiGroupDeletedIndication NOTIFICATION-TYPE definition. The associated data varbinds are assigned from the indication parameters as follows:

1. The first varbind is an instance of dmiCompId.
2. The second varbind is an instance of dmiGroupId

5.7.6 Mapping DMI Language Added Indication

Upon receipt of a DMI Language Added indication, construct an SNMPv2 Trap as specified by the dmiLanguageAddedIndication NOTIFICATION-TYPE definition. The associated data varbinds are assigned from the indication parameters as follows:

1. The first varbind is an instance of dmiCompId.
2. The second varbind is an instance of dmiLanguage.

5.7.7 Mapping DMI Language Deleted Indication

Upon receipt of a DMI Language Deleted indication, construct an SNMPv2 Trap as specified by the dmiLanguageDeletedIndication NOTIFICATION-TYPE definition. The associated data varbinds are assigned from the indication parameters as follows:

1. The first varbind is an instance of dmiCompId.
2. The second varbind is an instance of dmiLanguage.

5.7.8 Processing DMI Subscription Notice Indication

The DMI Subscription Notice indication is NOT translated into an SNMP Trap. This indication is only sent to the subscriber associated with the subscription entry. Therefore, it is not relevant to other indication and event recipients.

Upon receipt of a DMI Subscription Notice indication, the mapping agent should take appropriate measures to re-subscribe with the DMI Service Provider so that DMI Event and Indication reception is not interrupted.

6 The DMTF-DMI-MIB definitions

The DMTF-DMI-MIB is an integral part of the mapping scheme defined in this document. Its implementation is mandatory for compliance with this standard.

The dmiClasses group provides complete access to the DMI meta-data for all registered group classes. This information can be retrieved via SNMP and used to perform the MIF to MIB conversion process defined in this document. A savvy MIB Browser could even use the information directly to perform the same functions that it could if all the converted-MIF (mapped) MIBs had been compiled into the management system.

The dmiComponents group provides the containment information that is lost in the mapping scheme; i.e., which group instances are present in each component. It also provides some component level data which is not present in the ComponentID group

The dmiTraps and dmiTrapVars groups define traps that correspond to the standard DMI indications. Event indications are mapped into specific traps defined in the individual MIFs to MIB translations.

The DMTF-DMI-MIB also provides definitions for dmtfStdMifs, dmtfDynMifs, dmiCompId, dmiGroupId, DmiDate, DmiInteger64, and DmiString. They all can be imported into the other translated MIBs as necessary.

7 References

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

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

[3] Case, J., Fedor, M., Schoffstall, M., Davin, J., "Simple Network Management Protocol", STD 15, RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[4] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Introduction to Community-based SNMPv2", RFC 1901, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[5] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1902, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[6] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1903, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[7] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1904, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[8] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[9] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[10] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[11] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Coexistence Between Version 1 and Version 2 of the Internet-standard Network Framework", RFC 1908, SNMP Research, Inc., cisco Systems, Inc, Dover Beach Consulting, Inc., International Network Services, January 1996.

[12] Desktop Management Task Force, "Desktop Management Interface Specification", version 1.0, September 1994.

[13] Desktop Management Task Force, "Desktop Management Interface Reference",

version 1.1, January 1996.

[14] Desktop Management Task Force, "Desktop Management Interface Specification", version 2.0, March 1996.

[15] Desktop Management Task Force, "LAN Adapter Standard Groups Definition", version 1.0, October 1994

[16] Desktop Management Task Force, "Software Standard MIF Definition", version 2.0, November 1995[17] Desktop Management Task Force, "PC Systems Standard MIF Definition", version 1.3, March 1995.

[18] Desktop Management Task Force, "Systems Standard Group Definitions", version 1.0, May 1996.

[19] Desktop Management Task Force, "Large Mailing Organization MIF Definition", version 1.0, April 1995.

[20] Desktop Management Task Force, "Monitor MIF Definition", version 1.0, January 1996.

[21] Desktop Management Task Force, "Printer MIF Definition", version 8.4, March 1995.

8 Acknowledgements

The authors wish to acknowledge the contributions of the DMTF DMI to SNMP Mapping Working Committee in general. Special thanks is extended to Asher Vardi and Sasha Tolpin of the Intel Architecture Lab for their comments and implementation experience of this specification.

9 Security Considerations

This specification raises no security issues beyond those currently associated with DMI 2.0 and SNMPv1/v2C. It is expected that as security methods are standardized for both DMI and SNMP, the security methods will need to be implemented by the SNMP-to-DMI mapping agent to operate sucessfully.

10 Authors' Address

Brian O'Keefe
Hewlett-Packard Company, Inc.
3404 East Harmony Road
Fort Collins, CO 80528

Voice: 970-868-4303 (970-229-4303 until 12/31/97)
Fax: 970-868-2038
Email: brian_okeefe@hp.com
bok@cnd.hp.com

Steve Bostock
Novell, Inc.
2180 Fortune Drive
San Jose, CA 95131

Voice: 408-577-8203
Fax: 408-577-5706
Email: Steve_Bostock@novell.com
steveb@novell.com