

1.	Introduction.....	2
2.	Application Program Interfaces.....	2
2.1.	Extension Agent APIs.....	3
2.1.1.	SnmpExtensionInit().....	3
2.1.2.	SnmpExtensionQuery().....	4
2.1.3.	SnmpExtensionTrap().....	6
2.2.	Manager APIs.....	7
2.2.1.	SnmpMgrOpen().....	7
2.2.2.	SnmpMgrClose().....	9
2.2.3.	SnmpMgrRequest().....	10
2.2.4.	SnmpMgrStrToOid().....	11
2.2.5.	SnmpMgrOidToStr().....	12
2.2.6.	SnmpMgrTrapListen().....	13
2.2.7.	SnmpMgrGetTrap().....	14
2.3.	Utility APIs.....	15
2.3.1.	SnmpUtilOidCpy().....	15
2.3.2.	SnmpUtilOidAppend().....	16
2.3.3.	SnmpUtilOidNCmp().....	17
2.3.4.	SnmpUtilOidCmp().....	18
2.3.5.	SnmpUtilOidFree().....	19
2.3.6.	SnmpUtilVarBindListCpy().....	20
2.3.7.	SnmpUtilVarBindCpy().....	21
2.3.8.	SnmpUtilVarBindListFree().....	22
2.3.9.	SnmpUtilVarBindFree().....	23
2.3.10.	SnmpUtilPrintAsnAny().....	24
3.	Structures and Types.....	25
3.1.	Structures.....	25
3.1.1.	AsnAny.....	25
3.1.2.	RFC1157VarBind.....	26
3.1.3.	RFC1157VarBindList.....	26
3.2.	Types.....	27
3.2.1.	AsnInteger.....	27
3.2.2.	AsnOctetString.....	27
3.2.3.	AsnObjectIdentifier.....	27
3.2.4.	AsnSequence.....	27
3.2.5.	AsnImplicitSequence.....	27
3.2.6.	AsnIPAddress.....	27
3.2.7.	AsnDisplayString.....	28
3.2.8.	AsnCounter.....	28
3.2.9.	AsnGauge.....	28
3.2.10.	AsnTimeticks.....	28
3.2.11.	AsnOpaque.....	28
4.	MIB Compiler.....	29
5.	Example Code.....	30
5.1.	Example Extension Agent.....	30
5.1.1.	TESTDLL.DEF.....	30
5.1.2.	TESTDLL.C.....	30
5.1.3.	TESTMIB.C.....	36
5.1.4.	TESTMIB.H.....	48
5.2.	Example Manager Application.....	51
5.2.1.	SNMPUTIL.C.....	51

6.	References.....	60
7.	Revision History.....	60

Microsoft Windows/NT SNMP Programmer's Reference

Note:

This PRE-RELEASE documentation contains information that may change prior to the official release of Windows/NT. Microsoft makes no warranty of any kind as to the suitability or correctness of this document.

July 22, 1992

1. Introduction

This document presents and describes the Application Program Interfaces (APIs) supporting Simple Network Management Protocol (SNMP) [1,2] for Windows/NT.

These APIs support the development of SNMP agent applications and SNMP manager applications. A SNMP agent application is a SNMP application entity that responds to queries from SNMP manager applications and generates traps to SNMP manager applications. A SNMP manager application is a SNMP application entity that generates queries to SNMP agent applications and receives traps generated by SNMP agent applications.

The SNMP agent is extendible using an agent multiplexing design - a given Protocol Data Unit's (PDU) variable bindings can refer to Management Information Base (MIB) variables supported by many Extension Agents. It consists of the Microsoft Extendible Agent; and Microsoft and Independent Software Vendor (ISV) developed Extension Agent modules that interact with the Extendible Agent. Microsoft will initially implement the following MIBs as Extension Agents: RFC1156 [3], LanMgr-Mib-II [4], and LanMgr-Alerts-II [5]. The Extendible Agent is implemented as a Win32 Service, and the Extension Agents are implemented as Win32 Dynamic Link Libraries (DLLs).

The Manager APIs allow multiple managers to simultaneously coexist. ISV developed manager applications access the Manager API to perform SNMP manager operations. The Manager APIs are implemented as a Win32 DLL, and a single detached process. These interact with one or more ISV developed manager applications.

Miscellaneous utility APIs are also provided to assist with comparing, copying, and freeing allocated data structures, among other operations.

2. Application Program Interfaces

There are three major categories of APIs provided: Agent, Manager, and Utility.

Wherever possible, the low-level details of SNMP have been hidden by these APIs. Abstract Syntax Notation 1 (ASN.1) and its Basic Encoding Rules (BER) are not exposed by these APIs. The formatting and parsing of SNMP packets, and the communications code that communicates SNMP packets on the network are not exposed by these APIs.

The APIs utilize typedefs for SNMP variables, SNMP variable bindings, and SNMP variable bindings lists. These definitions along with the API parameters comprise the interface exposed to the ISV developing a SNMP agent or manager application.

Agent APIs are provided to define the interface between the Extendible Agent and the ISV developed Extension Agent DLLs.

Manager APIs are provided to define the interface between ISV developed Manager Applications and the Management API DLL.

Utility APIs are provided to simplify manipulation of the typedefs discussed above, and perform other miscellaneous operations.

This results in the SNMP developer being able to concentrate on the task of providing or requesting management information without having to be concerned with socket programming, message generation and parsing, ASN.1 BER encoding and decoding, and other low-level details of SNMP. This greatly simplifies the task of developing SNMP agent and manager applications.

2.1. Extension Agent APIs

Agent APIs are provided to define the interface between the Extendible Agent and the ISV developed Extension Agent DLLs.

These APIs are implemented by the ISV developing the Extension Agent DLL, and are called by the Extendible Agent provided by Microsoft. Some may view these as similar to callbacks. No active thread of execution is required in an Extension Agent DLL, but may be implemented if necessary to provided the desired functionality (note: none of the Microsoft Extension Agent DLLs discussed earlier contain an active thread of execution).

These ISV developed Extension Agent DLLs are dynamically linked by the Extendible Agent at run-time. The Extendible Agent Service determines what Extension Agent DLLs need to be loaded by referring to its parameters in the Registry Database.

There are three Extension Agent APIs: `SnmpExtensionInit()`, `SnmpExtensionQuery()`, and `SnmpExtensionTrap()`. `SnmpExtensionTrap()` could be considered optional if the Extension Agent indicates it will not be generating traps when `SnmpExtensionInit()` is called.

2.1.1. SnmpExtensionInit()

```

BOOL SnmpExtensionInit(
    IN  DWORD          dwTimeZeroReference,
    OUT HANDLE         *hPollForTrapEvent,
    OUT AsnObjectIdentifier *supportedView)

```

The **SnmpExtensionInit** function in the Extension Agent DLL is called by the Extendible Agent to perform bi-lateral initialization of both Extension and Extendible Agents.

Parameter	Description
<code>dwTimeZeroReference</code>	Specifies a time-zero reference for the Extension Agent.
<code>hPollForTrapEvent</code>	Points to an Event handle for an event that will be asserted when the SnmpExtensionTrap entry point should be polled by the Extendible Agent. If traps are not generated, NULL should be returned.
<code>supportedView</code>	Points to an <code>AsnObjectIdentifier</code> specifying the MIB sub-tree supported by the Extension Agent.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

The `dwTimeZeroReference` allows all Extension Agents to report time information from the same reference point. The Extension Agent can compute elapsed time by subtracting `dwTimeZeroReference` from the value returned by **GetCurrentTime**. This time reference is necessary to implement traps, and possibly some MIB variables.

The Extension Agent can generate traps in a variety of ways. The `hPollForTrapEvent` is provided to allow this generality. It is created by the Extension Agent by using **CreateEvent**. When this event is asserted, the Extendible Agent will call the **SnmpExtensionTrap** entry point. This event can be asserted many ways, **SetEvent** for example.

See Also

`SnmpExtensionTrap()`, `GetCurrentTime()`, `CreateEvent()`, `SetEvent()`.

2.1.2. SnmpExtensionQuery()

```

BOOL SnmpExtensionQuery(
    IN BYTE                requestType,
    IN OUT RFC1157VarBindList *variableBindings,
    OUT AsnInteger          *errorStatus,
    OUT AsnInteger          *errorIndex)

```

The **SnmpExtensionQuery** function in the Extension Agent DLL is called by the Extendible Agent to resolve SNMP requests.

Parameter	Description
requestType	Specifies the SNMP request type from the following list:
requestType	Description
ASN_RFC1157_GETREQUEST	Indicates SNMP Get Request.
ASN_RFC1157_GETNEXTREQUEST	Indicates SNMP Get Next Request.
ASN_RFC1157_SETREQUEST	Indicates SNMP Set Request.
variableBindings	Points to the variable bindings list.
errorStatus	Points to variable to receive the resulting error status from the following list (see sections 4.1.2, 4.1.3, and 4.1.5 of RFC1157 [2] to understand the meaning of these errors for the supplied requestType):
errorStatus	Description
ASN_ERRORSTATUS_NOERROR	Indicates the noError error status.
ASN_ERRORSTATUS_TOOBIG	Indicates the tooBig error status.
ASN_ERRORSTATUS_NOSUCHNAME	Indicates the noSuchName error status.
ASN_ERRORSTATUS_BADVALUE	Indicates the badValue error status.
ASN_ERRORSTATUS_READONLY	Indicates the readOnly error status.
ASN_ERRORSTATUS_GENERR	Indicates the genError error status.
errorIndex	Points to variable to receive the resulting error index.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

This entry point is called when the Extendible Agent must resolve a Get, Get Next, or Set operation within the subtree indicated by **SnmpExtensionInit**. For Get or Set, the implementor must follow the rules in RFC 1157 [2] to resolve the variable bindings or generate an error.

Get Next is more complicated to process. For Get Next, the requested variable may not be resolvable by this Extension Agent. If it can be resolved by this Extension Agent, the implementor must follow the rules in RFC 1157 [2] to resolve the variable bindings or generate an error. If it cannot be resolved by this Extension Agent, the implementor must alter the name field of the variable binding to point just past the supportedView provided by **SnmpExtensionInit**. If the agent's supported view was ".1.3.6.1.4.1.77.1", a get next on ".1.3.6.1.4.1.77.1.5.1" would result in the name field being modified to be ".1.3.6.1.4.1.77.2". This signals the Extendible Agent to continue to attempt to resolve such variable bindings with other Extension Agents.

See Also

SnmpExtensionInit(), RFC 1157.

2.1.3. SnmpExtensionTrap()

```

BOOL SnmpExtensionTrap(
    OUT AsnObjectIdentifier *enterprise,
    OUT AsnInteger           *genericTrap,
    OUT AsnInteger           *specificTrap,
    OUT AsnTimeticks         *timeStamp,
    OUT RFC1157VarBindList   *variableBindings)

```

The **SnmpExtensionTrap** function in the Extension Agent DLL is called by the Extendible Agent to retrieve Extension Agent generated traps.

Parameter	Description																
enterprise	Points to an object identifier indicating the originating enterprise generating the trap.																
genericTrap	Points to an indication of the generic trap generated from the following list:																
<table> <tr> <th>genericTrap</th><th>Description</th></tr> <tr> <td>SNMP_GENERICTRAP_COLDSTART</td><td>Indicates Cold Start trap.</td></tr> <tr> <td>SNMP_GENERICTRAP_WARMSTART</td><td>Indicates Warm Start trap.</td></tr> <tr> <td>SNMP_GENERICTRAP_LINKDOWN</td><td>Indicates Link Down trap.</td></tr> <tr> <td>SNMP_GENERICTRAP_LINKUP</td><td>Indicates Link Up trap.</td></tr> <tr> <td>SNMP_GENERICTRAP_AUTHFAILURE</td><td>Indicates Authentication Failure trap.</td></tr> <tr> <td>SNMP_GENERICTRAP_EGPNEIGHLOSS</td><td>Indicates EGP Neighbor Loss trap.</td></tr> <tr> <td>SNMP_GENERICTRAP_ENTERSPECIFIC</td><td>Indicates Enterprise Specific trap.</td></tr> </table>		genericTrap	Description	SNMP_GENERICTRAP_COLDSTART	Indicates Cold Start trap.	SNMP_GENERICTRAP_WARMSTART	Indicates Warm Start trap.	SNMP_GENERICTRAP_LINKDOWN	Indicates Link Down trap.	SNMP_GENERICTRAP_LINKUP	Indicates Link Up trap.	SNMP_GENERICTRAP_AUTHFAILURE	Indicates Authentication Failure trap.	SNMP_GENERICTRAP_EGPNEIGHLOSS	Indicates EGP Neighbor Loss trap.	SNMP_GENERICTRAP_ENTERSPECIFIC	Indicates Enterprise Specific trap.
genericTrap	Description																
SNMP_GENERICTRAP_COLDSTART	Indicates Cold Start trap.																
SNMP_GENERICTRAP_WARMSTART	Indicates Warm Start trap.																
SNMP_GENERICTRAP_LINKDOWN	Indicates Link Down trap.																
SNMP_GENERICTRAP_LINKUP	Indicates Link Up trap.																
SNMP_GENERICTRAP_AUTHFAILURE	Indicates Authentication Failure trap.																
SNMP_GENERICTRAP_EGPNEIGHLOSS	Indicates EGP Neighbor Loss trap.																
SNMP_GENERICTRAP_ENTERSPECIFIC	Indicates Enterprise Specific trap.																
specificTrap	Points to an indication of the specific trap generated.																
timeStamp	Points to variable to receive the time-stamp.																
variableBindings	Points to the variable bindings list.																

Returns

If the function returned a trap in its parameters, the return value is TRUE. The function is called repeatedly by the Extendible Agent until a return value of FALSE is returned.

Comments

This function is repeatedly called by the Extendible Agent once hPollForTrapEvent has been asserted. Each successful call returns a single trap's data in its parameters. It returns FALSE to indicate that its parameters do not represent valid trap data and to stop the Extendible Agent's repeated calls.

See Also

SnmpExtensionInit().

2.2. Manager APIs

Manager APIs are provided to define the interface between ISV developed Manager Applications and the Management API DLL.

These APIs are provided by the Manager API DLL. Internal to the Manager API, a detached process is implemented to receive SNMP traps and dispatch them to the Manager Applications that have registered to receive SNMP traps (using `SnmpMgrTrapListen()`). The Manager APIs are synchronous excepting the APIs dealing with SNMP traps, which are either polled or notification based.

The ISV developed Manager Application links with this DLL to gain access to its services.

There are several Manager APIs: `SnmpMgrOpen()`, `SnmpMgrClose()`, `SnmpMgrRequest()`, `SnmpMgrStrToOid()`, `SnmpMgrOidToStr()`, `SnmpMgrTrapListen()`, and `SnmpMgrGetTrap()`. `SnmpMgrOpen()`, `SnmpMgrClose()`, and `SnmpMgrRequest()` are the primary APIs. `SnmpMgrStrToOid()` and `SnmpMgrOidToStr()` are only accessed if their respective conversions of Object Identifiers to/from Object Descriptors are desired. `SnmpMgrTrapListen()` and `SnmpMgrGetTrap()` are only accessed if trap reception is desired.

2.2.1. SnmpMgrOpen()

LPSNMP_MGR_SESSION `SnmpMgrOpen()`

```
IN LPSTR lpAgentAddress,    // Name/address of target SNMP agent
IN LPSTR lpAgentCommunity, // Community for target SNMP agent
IN INT   nTimeOut,         // Communication time-out in milliseconds
IN INT   nRetries          // Communication time-out/retry count
```

The **SnmpMgrOpen** function initializes communications sockets and data structures allowing communications with the specified agent.

Parameter	Description
<code>lpAgentAddress</code>	Points to a null-terminated string specifying either a dotted-decimal IP address, or a host name that can be resolved to an IP address.
<code>lpAgentCommunity</code>	Points to a null-terminated string specifying the SNMP Community Name used when communicating with the agent specified in <code>lpAgentAddress</code> .
<code>nTimeOut</code>	Specifies the communications time-out in milliseconds.
<code>nRetries</code>	Specifies the communications retry count. The time-out specified in <code>nTimeOut</code> is doubled each time a retry attempt is transmitted.

Returns

If the function is successful, the return value is a pointer to a `LPSNMP_MGR_SESSION` structure. This structure is used internally and should not be altered by the programmer.

If the function fails, the return value is `NULL`. Use the **GetLastError** function to obtain extended error information.

<code>GetLastError()</code>	Description
<code>SNMP_MEM_ALLOC_ERROR</code>	Indicates error allocating memory.

Comments

None.

See Also

`SnmpMgrClose()`, `SnmpMgrRequest()`.

2.2.2. SnmpMgrClose()

BOOL SnmpMgrClose(

IN LPSNMP_MGR_SESSION session) // SNMP session pointer

The **SnmpMgrClose** function closes communications socket and data structures associated with the specified session.

Parameter	Description
session	Points to a LPSNMP_MGR_SESSION structure specifying the session to close.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE. Use the **GetLastError** function to obtain extended error information.

Comments

None.

See Also

SnmpMgrOpen(), SnmpMgrRequest().

2.2.3. SnmpMgrRequest()

SNMPAPI SnmpMgrRequest(

```
IN    LPSNMP_MGR_SESSION session,           // SNMP session pointer
IN    BYTE                requestType,       // Get, GetNext, or Set
IN OUT RFC1157VarBindList *variableBindings, // Variable bindings
OUT    AsnInteger          *errorStatus,     // Result error status
OUT    AsnInteger          *errorIndex)      // Result error index
```

The **SnmpMgrRequest** function requests the specified operation be performed with the specified agent.

Parameter	Description
session	Points to a LPSNMP_MGR_SESSION structure specifying the session that will perform the request.
requestType	Specifies the SNMP request type from the following list:

requestType	Description
ASN_RFC1157_GETREQUEST	Indicates SNMP Get Request.
ASN_RFC1157_GETNEXTREQUEST	Indicates SNMP Get Next Request.
ASN_RFC1157_SETREQUEST	Indicates SNMP Set Request.

variableBindings	Points to the variable bindings list.
errorStatus	Points to variable to receive the resulting error status from the following list (see sections 4.1.2, 4.1.3, and 4.1.5 of RFC 1157 [2] to understand the meaning of these errors for the supplied requestType):

errorStatus	Description
ASN_ERRORSTATUS_NOERROR	Indicates the noError error status.
ASN_ERRORSTATUS_TOOBIG	Indicates the tooBig error status.
ASN_ERRORSTATUS_NOSUCHNAME	Indicates the noSuchName error status.
ASN_ERRORSTATUS_BADVALUE	Indicates the badValue error status.
ASN_ERRORSTATUS_READONLY	Indicates the readOnly error status.
ASN_ERRORSTATUS_GENERR	Indicates the genError error status.

errorIndex	Points to variable to receive the resulting error index.
------------	--

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is NULL. Use the **GetLastError** function to obtain extended error information.

GetLastError()	Description
SNMP_MGMTAPI_TIMEOUT	Request timed-out.
SNMP_MGMTAPI_SELECT_FDERRORS	Unexpected error file descriptors indicated by select().

Comments

None.

See Also

SnmpMgrOpen(), SnmpMgrClose(), RFC 1157.

2.2.4. SnmpMgrStrToOid()

```
BOOL SnmpMgrStrToOid(  
    IN LPSTR          string,    // OID string to be converted  
    OUT AsnObjectIdentifier *oid) // OID internal representation
```

The **SnmpMgrStrToOid** function converts a string OBJECT IDENTIFIER or OBJECT DESCRIPTOR representation to an internal AsnObjectIdentifier.

Parameter	Description
string	Points to a NULL terminated string to be converted.
oid	Points to an AsnObjectIdentifier variable to receive the converted value.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

None.

See Also

SnmpMgrOidToStr(), MibCC.

2.2.5. SnmpMgrOidToStr()

```
BOOL SnmpMgrOidToStr(  
    OUT AsnObjectIdentifier oid,    // OID internal rep to be converted  
    IN  LPSTR                    string) // OID string representation
```

The **SnmpMgrStrToOid** function converts an internal AsnObjectIdentifier to a string OBJECT IDENTIFIER or OBJECT DESCRIPTOR representation.

Parameter	Description
oid	Points to an AsnObjectIdentifier to be converted.
string	Points to a NULL terminated string to receive the converted value.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

None.

See Also

SnmpMgrStrToOid(), MibCC.

2.2.6. SnmpMgrTrapListen()

BOOL SnmpMgrTrapListen(

OUT HANDLE *phTrapAvailable) // Event handle indicating trap(s) available

The **SnmpMgrTrapListen** function registers the desire to receive SNMP traps.

Parameter	Description
phTrapAvailable	Points to an event that is asserted when new traps have been received.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE. Use the **GetLastError** function to obtain extended error information.

GetLastError()	Description
SNMP_MGMTAPI_TRAP_DUPINIT	This function has already been called.
SNMP_MGMTAPI_AGAIN	Error encountered, can attempt call again.

Comments

The phTrapAvailable event is provided to allow event-driven acquisition of SNMP traps. It can be ignored and the **SnmpMgrGetTrap** function can be polled at regular intervals.

Or, a thread could be created waiting on the event using **WaitForSingleObject**. When the event is asserted, this thread should clear the event using **ResetEvent**, and then repeatedly call the **SnmpMgrGetTrap** function until it returns FALSE.

See Also

SnmpMgrGetTrap(), WaitForSingleObject(), ResetEvent().

2.2.7. SnmpMgrGetTrap()

```

BOOL SnmpMgrGetTrap(
    OUT AsnObjectIdentifier *enterprise,    // Generating enterprise
    OUT AsnInteger          *genericTrap,   // Generic trap type
    OUT AsnInteger          *specificTrap,   // Enterprise specific type
    OUT AsnTimeticks        *timeStamp,     // Time stamp
    OUT RFC1157VarBindList  *variableBindings // Variable bindings

```

The **SnmpMgrGetTrap** function returns outstanding trap data that the caller has not yet received.

Parameter	Description
enterprise	Points to an object identifier indicating the originating enterprise generating the trap.
genericTrap	Points to an indication of the generic trap generated from the following list:
genericTrap	Description
SNMP_GENERICTRAP_COLDSTART	Indicates Cold Start trap.
SNMP_GENERICTRAP_WARMSTART	Indicates Warm Start trap.
SNMP_GENERICTRAP_LINKDOWN	Indicates Link Down trap.
SNMP_GENERICTRAP_LINKUP	Indicates Link Up trap.
SNMP_GENERICTRAP_AUTHFAILURE	Indicates Authentication Failure trap.
SNMP_GENERICTRAP_EGPNEIGHLOSS	Indicates EGP Neighbor Loss trap.
SNMP_GENERICTRAP_ENTERSPECIFIC	Indicates Enterprise Specific trap.
specificTrap	Points to an indication of the specific trap generated.
timeStamp	Points to variable to receive the time-stamp.
variableBindings	Points to the variable bindings list.

Returns

If the function returned a trap in its parameters, the return value is TRUE. The function should be called repeatedly until a return value of FALSE is returned with **GetLastError()** indicating either an error or SNMP_MGMTAPI_NOTRAPs.

GetLastError()	Description
SNMP_MGMTAPI_TRAP_ERRORS	Errors encountered, traps not accessible.
SNMP_MGMTAPI_NOTRAPs	No traps are available.

Comments

The phTrapAvailable event from **SnmpMgrTrapListen** is provided to allow event-driven acquisition of SNMP traps. It can be ignored and this function can be polled at regular intervals.

Or, a thread could be created waiting on the event using **WaitForSingleObject**. When the event is asserted, this thread should clear the event using **ResetEvent**, and then repeatedly call this function until it returns FALSE.

See Also

SnmpMgrTrapListen(), WaitForSingleObject(), ResetEvent().

2.3. Utility APIs

Utility APIs are provided to simplify manipulation of the typedefs discussed above, and perform other miscellaneous operations.

These APIs are provided by the common SNMP library, **SNMP.LIB**.

The ISV developed Application links with this library to gain access to its services.

There are several Utility APIs: `SnmUtilOidCpy()`, `SnmUtilOidAppend()`, `SnmUtilOidNCmp()`, `SnmUtilOidCmp()`, `SnmUtilOidFree()`, `SnmUtilVarBindListCpy()`, `SnmUtilVarBindCpy()`, `SnmUtilVarBindListFree()`, `SnmUtilVarBindFree()`, and `SnmUtilPrintAsnAny()`.

Care should be taken to insure that dynamic memory allocated for SNMP data structures is freed using the appropriate API. These APIs are *smart* and know how to free other data contained in these data structures.

2.3.1. `SnmUtilOidCpy()`

```
SNMPAPI SnmUtilOidCpy(
    OUT AsnObjectIdentifier *DestObjId, // Destination OID
    IN AsnObjectIdentifier *SrcObjId    // Source OID
);
```

The **`SnmUtilOidCpy`** function copies the `SrcObjId` to the `DestObjId` allocating any necessary memory for the destination's copy.

Parameter	Description
<code>DestObjId</code>	Points to an <code>AsnObjectIdentifier</code> variable to receive the copy.
<code>SrcObjId</code>	Points to an <code>AsnObjectIdentifier</code> variable to copy.

Returns

If the function is successful, the return value is `TRUE`.

If the function fails, the return value is `FALSE`.

Comments

Any memory allocated for the destination can be freed using **`SnmUtilOidFree`**.

See Also

`SnmUtilOidFree()`.

2.3.2. SnmpUtilOidAppend()

```
SNMPAPI SnmpUtilOidAppend(
    IN OUT AsnObjectIdentifier *DestObjId, // Destination OID
    IN AsnObjectIdentifier *SrcObjId      // Source OID
);
```

The **SnmpUtilOidAppend** function appends the SrcObjId to the DestObjId reallocating any necessary memory for the destination's copy.

Parameter	Description
DestObjId	Points to an AsnObjectIdentifier variable to receive the copy.
SrcObjId	Points to an AsnObjectIdentifier variable to copy.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

Any memory allocated for the destination can be freed using **SnmpUtilOidFree**.

See Also

SnmpUtilOidFree().

2.3.3. SnmpUtilOidNCmp()

```
int SnmpUtilOidNCmp(
    IN AsnObjectIdentifier *A, // First OID
    IN AsnObjectIdentifier *B, // Second OID
    IN UINT Len                // Max len to compare
);
```

The **SnmpUtilOidNCmp** function compares Len sub-identifiers of A and B.

Parameter	Description
A	Points to an AsnObjectIdentifier variable to compare.
B	Points to an AsnObjectIdentifier variable to compare.
Len	Indicates the number of sub-identifiers to compare.

Returns

The function returns a value greater than zero if A is greater than B, zero if A equals B, and less than zero if A is less than B.

Comments

None.

See Also

SnmpUtilOidCmp().

2.3.4. SnmpUtilOidCmp()

```
int SnmpUtilOidCmp(  
    IN AsnObjectIdentifier *A, // First OID  
    IN AsnObjectIdentifier *B, // Second OID  
);
```

The **SnmpUtilOidCmp** function compares object identifiers A and B.

Parameter	Description
A	Points to an AsnObjectIdentifier variable to compare.
B	Points to an AsnObjectIdentifier variable to compare.

Returns

The function returns a value greater than zero if A is greater than B, zero if A equals B, and less than zero if A is less than B.

Comments

None.

See Also

SnmpUtilOidNCmp().

2.3.5. SnmpUtilOidFree()

```
void SnmpUtilOidFree(
    IN OUT AsnObjectIdentifier *Obj // OID to free
);
```

The **SnmpUtilOidFree** function frees any allocated data associated with Obj.

Parameter	Description
Obj	Points to an AsnObjectIdentifier variable whose allocated data should be freed.

Returns

None.

Comments

None.

See Also

None.

2.3.6. SnmpUtilVarBindListCpy()

```
SNMPAPI SnmpUtilVarBindListCpy(
    RFC1157VarBindList *dst, // Destination var bind list
    RFC1157VarBindList *src  // Source var bind list
);
```

The **SnmpUtilOidCpy** function copies the RFC1157VarBindList allocating any necessary memory for the destination's copy.

Parameter	Description
dst	Points to an RFC1157VarBindList to receive the copy.
src	Points to an RFC1157VarBindList to copy.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

Any memory allocated for the destination can be freed using **SnmpUtilVarBindListFree**.

See Also

SnmpUtilVarBindListFree().

2.3.7. SnmpUtilVarBindCpy()

```
SNMPAPI SnmpUtilVarBindCpy(  
    RFC1157VarBind *dst, // Destination var bind  
    RFC1157VarBind *src  // Source var bind  
);
```

The **SnmpUtilOidCpy** function copies the RFC1157VarBind allocating any necessary memory for the destination's copy.

Parameter	Description
dst	Points to an RFC1157VarBind to receive the copy.
src	Points to an RFC1157VarBind to copy.

Returns

If the function is successful, the return value is TRUE.

If the function fails, the return value is FALSE.

Comments

Any memory allocated for the destination can be freed using **SnmpUtilVarBindFree**.

See Also

SnmpUtilVarBindFree().

2.3.8. SnmpUtilVarBindListFree()

```
void SnmpUtilVarBindListFree(
    RFC1157VarBindList *VarBindList // Variable bindings list to free
);
```

The **SnmpUtilVarBindListFree** function frees any allocated data associated with VarBindList.

Parameter	Description
VarBindList	Points to an RFC1157VarBindList whose allocated data should be freed.

Returns

None.

Comments

None.

See Also

None.

2.3.9. SnmpUtilVarBindFree()

```
void SnmpUtilVarBindFree(
    RFC1157VarBind *VarBind // Variable binding to free
);
```

The **SnmpUtilVarBindFree** function frees any allocated data associated with VarBind.

Parameter	Description
VarBindList	Points to an RFC1157VarBind whose allocated data should be freed.
Returns	
None.	
Comments	
None.	
See Also	
None.	

2.3.10. SnmpUtilPrintAsnAny()

```
void SnmpUtilPrintAsnAny(  
    IN AsnAny *Any  
);
```

The **SnmpUtilPrintAsnAny** function prints the value of Any to stdout.

Parameter	Description
Any	Points to an AsnAny structure whose value is to be printed.

Returns

None.

Comments

This function is provided for debugging and development purposes. It does not generally print the data in forms a Manager Applications would normally need.

This function determines the type of data from the AsnAny structure which was probably set internally by referring to a BER tag value in an encoded stream.

See Also

AsnAny.

3. Structures and Types

3.1. Structures

3.1.1. AsnAny

```
typedef struct {
    BYTE asnType;
    union {
        // RFC 1155 SimpleSyntax (subset of ISO ASN.1)
        AsnInteger      number;
        AsnOctetString  string;
        AsnObjectIdentifier object;

        // ISO ASN.1
        AsnSequence      sequence;

        // RFC 1155 ApplicationSyntax
        AsnIPAddress      address;
        AsnCounter        counter;
        AsnGauge          gauge;
        AsnTimeticks      ticks;
        AsnOpaque         arbitrary;
    } asnValue;
} AsnAny;
```

The **AsnAny** structure contains a SNMP variable type and value. This structure is a member of the RFC1157VarBind structure used as a parameter in many of the SNMP APIs.

Member	Description
asnType	Indicates the variable's type and what portion of the union should be used.
asnType	Description
ASN_INTEGER	Indicates integer variable.
ASN_OCTETSTRING	Indicates octet string variable.
ASN_OBJECTIDENTIFIER	Indicates object identifier variable.
ASN_SEQUENCE	Indicates ASN sequence variable.
ASN_RFC1155_IPADDRESS	Indicates IP address variable.
ASN_RFC1155_COUNTER	Indicates counter variable.
ASN_RFC1155_GAUGE	Indicates gauge variable.
ASN_RFC1155_TIMETICKS	Indicates timeticks variable.
ASN_RFC1155_OPAQUE	Indicates opaque variable.
ASN_RFC1213_DISPSTRING	Indicates display string variable.
asnValue	Contains the variable's value. This union supports the possible SNMP values.
Member	Description

number	Accesses integer variable.
string	Accesses octet string variable.
object	Accesses object identifier variable.
sequence	Accesses ASN sequence variable.
address	Accesses IP address variable.
counter	Accesses counter variable.
gauge	Accesses gauge variable.
ticks	Accesses timeticks variable.
arbitrary	Accesses opaque variable.

Comments

None.

See Also

RFC1157VarBind, RFC 1155.

3.1.2. RFC1157VarBind

```
typedef struct vb {
    AsnObjectName  name;
    AsnObjectSyntax value;
} RFC1157VarBind;
```

The **RFC1157VarBind** structure represents a SNMP variable binding as defined in RFC 1157.

Member	Description
name	Indicates the variable's name as an object identifier.
value	Contains the variable's value.

Comments

None.

See Also

RFC 1155, RFC 1157.

3.1.3. RFC1157VarBindList

```
typedef struct {
    RFC1157VarBind *list;
    UINT            len;
} RFC1157VarBindList;
```

The **RFC1157VarBindList** structure represents a SNMP variable bindings list as defined in RFC 1157.

Member	Description
list	A pointer that may be dereferenced as an array to access individual variable bindings.
len	Contains the number of variable bindings in the list.

Comments

None.

See Also

RFC1157VarBind, RFC 1157.

3.2. Types

3.2.1. AsnInteger

```
typedef long      AsnInteger;
```

AsnInteger is used to represent signed integer quantities as defined in RFC 1155.

See Also

RFC 1155.

3.2.2. AsnOctetString

```
typedef struct {  
    BYTE *stream;  
    UINT  length;  
    BOOL  dynamic;  
}          AsnOctetString;
```

AsnOctetString is used to represent octet (usually byte) quantities as defined in RFC 1155.

The dynamic flag indicates to data structure freeing code whether the stream is allocated from dynamic memory.

See Also

RFC 1155.

3.2.3. AsnObjectIdentifier

```
typedef struct {  
    UINT idLength;  
    UINT *ids;  
}          AsnObjectIdentifier;
```

AsnObjectIdentifier is used to represent objects as defined in RFC 1155.

See Also

RFC 1155.

3.2.4. AsnSequence

```
typedef AsnOctetString AsnSequence;
```

AsnSequence is used to represent sequences as defined in RFC 1155.

See Also

RFC 1155.

3.2.5. AsnImplicitSequence

```
typedef AsnSequence  AsnImplicitSequence;
```

AsnImplicitSequence is used to represent implicit sequences as defined in RFC 1155.

See Also

RFC 1155.

3.2.6. AsnIPAddress

`typedef AsnOctetString AsnIPAddress;`

AsnIPAddress is used to represent Internet Protocol (IP) addresses as defined in RFC 1155.

See Also

RFC 1155.

3.2.7. AsnDisplayString

typedef AsnOctetString AsnDisplayString;

AsnDisplayString is used to represent human readable octets as defined in RFC 1213.

See Also

RFC 1213.

3.2.8. AsnCounter

typedef AsnInteger AsnCounter;

AsnCounter is used to represent counted quantities as defined in RFC 1155.

See Also

RFC 1155.

3.2.9. AsnGauge

typedef AsnInteger AsnGauge;

AsnGauge is used to represent metered quantities as defined in RFC 1155.

See Also

RFC 1155.

3.2.10. AsnTimeticks

typedef AsnInteger AsnTimeticks;

AsnTimeTicks is used to represent relative time quantities as defined in RFC 1155.

See Also

RFC 1155.

3.2.11. AsnOpaque

typedef AsnOctetString AsnOpaque;

AsnOpaque is used to represent an encapsulation of other ASN constructs as defined in RFC 1155.

See Also

RFC 1155.

4. MIB Compiler

A basic SNMP MIB Compiler is provided with the Management APIs. This MIB compiler is currently used only to resolve the conversions requested by `SnmpMgrStrToOid()` and `SnmpMgrOidToStr()`. These conversions are Object Identifier to/from Object Descriptor translations.

Help on how this MIB compiler is invoked is provided by typing 'MIBCC -?':

4.1.\nt\release\i386\nt\system>

4.2.\nt\release\i386\nt\system>mibcc -?

usage: mibcc [-?] [-e] [-l] [-n] [-o] [-w] [files...]

MibCC compiles the specified SNMP MIB files.

- ? usage.
- eX stop after X Errors. (default = 10)
- l do not print Logo.
- n print each Node as it is added.
- ofile output file name. (default = mib.bin)
- wX set Warning level. (1=errors, 2=warnings)

4.3.\nt\release\i386\nt\system>

This MIB compiler accepts the subset of ASN.1 defined by the SNMP RFCs. A file containing ASN.1 source code can contain one or more MODULE definitions, although it is easier to keep one module per file and specify multiple files on the command line. Order of specification is significant. For example, SMI.MIB should be specified before any other MIBs.

An example compile may look like:

4.4.\nt\release\i386\nt\system>

4.5.\nt\release\i386\nt\system>mibcc smi.mib mib-ii.mib

Microsoft (R) SNMP MIB Compiler Version 1.00

Copyright (c) Microsoft Corporation 1992. All rights reserved.

warning : EXPORTS on line 3 not supported (ignored)

warning : IMPORTS on line 135 not supported (ignored)

Parse of 'test.mib' was successful. 2756 lines were parsed.

mibcc: total files processed: 1.

mibcc: writing compiled SNMP MIB.

The following MIB files have been included in this distribution: SMI.MIB, MIB-II.MIB, LMMIB2.MIB, LMALRT2.MIB, and TOASTER.MIB.

5. Example Code

Example SNMP application entities are contained in this section to aid in understanding the workings of the various APIs.

It should be noted that agent developers should thoroughly understand the non-protocol issues presented in RFC 1155 and RFC 1157.

5.1. Example Extension Agent

An example Extension Agent has been developed to illustrate the functionality of Win32 Extension Agents. This example agent implements the Toaster MIB that has been used as a simple example to illustrate SNMP.

This code illustrates how the ISV could implement the Agent APIs discussed above. It should be noted that a properly functioning agent must implement the SNMP operations correctly and that RFC 1157 should be consulted closely for information on how to resolve a query, what error status to return, etc.

For example, many agent developers make mistakes in implementing get next. This should be carefully considered in the development of Extension Agents under this architecture. Remember, a get next can occur at any arbitrary point in the OID space, including points where no variable exists.

5.1.1. TESTDLL.DEF

```
LIBRARY testdll
```

```
DESCRIPTION 'Sample SNMP Extension Agent for Windows NT.'
```

```
CODE LOADONCALL MOVEABLE DISCARDABLE
```

```
DATA PRELOAD MOVEABLE SINGLE
```

```
SEGMENTS
```

```
  _TEXT PRELOAD
```

```
  INIT_TEXT PRELOAD
```

```
HEAPSIZE 1024
```

```
EXPORTS
```

```
  SnmpExtensionInit
```

```
  SnmpExtensionTrap
```

```
  SnmpExtensionQuery
```

5.1.2. TESTDLL.C

```
/*++ BUILD Version: 0001    // Increment this if a change has global effects
```

```
Copyright (c) 1991 Microsoft Corporation
```

```
Module Name:
```

```
    testdll.c
```

```
Abstract:
```


Microsoft Windows/NT SNMP Programmer's Reference

Sample SNMP Extension Agent for Windows NT.

These files (testdll.c, testmib.c, and testmib.h) provide an example of how to structure an Extension Agent DLL which works in conjunction with the SNMP Extendible Agent for Windows NT.

Extensive comments have been included to describe its structure and operation. See also "Microsoft Windows/NT SNMP Programmer's Reference".

Created:

28-Jun-1991

Revision History:

--*/

```
static char *vcsid = "@(#) $Logfile:  N:/xtest/vcs/testdll.c_v  $ $Revision:  1.5  $";
```

```
// General notes:
```

```
//
```

```
// Microsoft's Extendible Agent for Windows NT is implemented by dynamically
// linking to Extension Agent DLLs that implement portions of the MIB.  These
// Extension Agents are configured in the Windows NT Registration Database.
// When the Extendible Agent Service is started, it queries the registry to
// determine which Extension Agent DLLs have been installed and need to be
// loaded and initialized.  The Extendible Agent invokes various DLL entry
// points (examples follow in this file) to request MIB queries and obtain
// Extension Agent generated traps.
```

```
// Necessary includes.
```

```
#include <windows.h>
```

```
#include <malloc.h>
```

```
#include <snmp.h>
```

```
// Contains definitions for the table structure describing the MIB.  This
// is used in conjunction with testmib.c where the MIB requests are resolved.
```

```
#include "testmib.h"
```

Microsoft Windows/NT SNMP Programmer's Reference

```
// Extension Agent DLLs need access to elapsed time agent has been active.  
// This is implemented by initializing the Extension Agent with a time zero  
// reference, and allowing the agent to compute elapsed time by subtracting  
// the time zero reference from the current system time. This example  
// Extension Agent implements this reference with dwTimeZero.
```

```
DWORD dwTimeZero = 0;
```

```
// Extension Agent DLLs that generate traps must create a Win32 Event object  
// to communicate occurrence of traps to the Extendible Agent. The event  
// handle is given to the Extendible Agent when the Extension Agent is  
// initialized, it should be NULL if traps will not be generated. This  
// example Extension Agent simulates the occurrence of traps with hSimulateTrap.
```

```
HANDLE hSimulateTrap = NULL;
```

```
// This is a standard Win32 DLL entry point. See the Win32 SDK for more  
// information on its arguments and their meanings. This example DLL does  
// not perform any special actions using this mechanism.
```

```
BOOL DllEntryPoint(  
    HANDLE hDll,  
    DWORD dwReason,  
    LPVOID lpReserved)  
{  
    switch(dwReason)  
    {  
        case DLL_PROCESS_ATTACH:  
        case DLL_PROCESS_DETACH:  
        case DLL_THREAD_ATTACH:  
        case DLL_THREAD_DETACH:  
        default:  
            break;  
    }  
    } // end switch()  
  
    return TRUE;  
  
} // end DllEntryPoint()
```

```
// Extension Agent DLLs provide the following entry point to coordinate the  
// initializations of the Extension Agent and the Extendible Agent. The  
// Extendible Agent provides the Extension Agent with a time zero reference;  
// and the Extension Agent provides the Extendible Agent with an Event handle  
// for communicating occurrence of traps, and an object identifier representing
```

Microsoft Windows/NT SNMP Programmer's Reference

```
// the root of the MIB subtree that the Extension Agent supports.

BOOL SnmpExtensionInit(
    IN  DWORD          dwTimeZeroReference,
    OUT HANDLE          *hPollForTrapEvent,
    OUT AsnObjectIdentifier *supportedView)
{
    // Record the time reference provided by the Extendible Agent.

    dwTimeZero = dwTimeZeroReference;

    // Create an Event that will be used to communicate the occurrence of traps
    // to the Extendible Agent. The Extension Agent will assert this Event
    // when a trap has occurred. This is explained further later in this file.

    if ((*hPollForTrapEvent = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)
    {
        // Indicate error?, be sure that NULL is returned to Extendible Agent.
    }

    // Indicate the MIB view supported by this Extension Agent, an object
    // identifier representing the sub root of the MIB that is supported.

    *supportedView = MIB_OidPrefix; // NOTE! structure copy

    // Record the trap Event. This example Extension Agent simulates traps by
    // generating a trap after every given number of processed requests.

    hSimulateTrap = *hPollForTrapEvent;

    // Indicate that Extension Agent initialization was successful.

    return TRUE;

} // end SnmpExtensionInit()

// Extension Agent DLLs provide the following entry point to communicate traps
// to the Extendible Agent. The Extendible Agent will query this entry point
// when the trap Event (supplied at initialization time) is asserted, which
// indicates that zero or more traps may have occurred. The Extendible Agent
// will repeatedly call this entry point until FALSE is returned, indicating
// that all outstanding traps have been processed.
```

```

BOOL SnmpExtensionTrap(
    OUT AsnObjectIdentifier *enterprise,
    OUT AsnInteger          *genericTrap,
    OUT AsnInteger          *specificTrap,
    OUT AsnTimeticks        *timeStamp,
    OUT RFC1157VarBindList  *variableBindings)
{
    // The body of this routine is an extremely simple example/simulation of
    // the trap functionality. A real implementation will be more complex.

    // The following define data inserted into the trap below. The Lan Manager
    // bytesAvailAlert from the Lan Manager Alerts-2 MIB is generated with an
    // empty variable bindings list.

    static UINT OidList[] = { 1, 3, 6, 1, 4, 1, 77, 2 };
    static UINT OidListLen = 8;

    // The following variable is used for the simulation, it allows a single
    // trap to be generated and then causes FALSE to be returned indicating
    // no more traps.

    static whichTime = 0;

    // The following if/else support the simulation.

    if (whichTime == 0)
    {
        whichTime = 1;    // Supports the simulation.

        // Communicate the trap data to the Extendible Agent.

        enterprise->idLength = OidListLen;
        enterprise->ids = (UINT *)malloc(sizeof(UINT) * OidListLen);
        memcpy(enterprise->ids, OidList, sizeof(UINT) * OidListLen);

        *genericTrap = SNMP_GENERICTRAP_ENTERSPECIFIC;

        *specificTrap = 1;           // the bytesAvailAlert trap

        *timeStamp = GetCurrentTime() - dwTimeZero;

        variableBindings->list = NULL;
        variableBindings->len = 0;
    }
}

```

```
// Indicate that valid trap data exists in the parameters.

return TRUE;
}
else
{
    whichTime = 0;    // Supports the simulation.

    // Indicate that no more traps are available and parameters do not
    // refer to any valid data.

    return FALSE;
}

} // end SnmpExtensionTrap()


// Extension Agent DLLs provide the following entry point to resolve queries
// for MIB variables in their supported MIB view (supplied at initialization
// time). The requestType is Get/GetNext/Set.

BOOL SnmpExtensionQuery(
    IN BYTE                requestType,
    IN OUT RFC1157VarBindList *variableBindings,
    OUT AsnInteger          *errorStatus,
    OUT AsnInteger          *errorIndex)
{
    static unsigned long requestCount = 0; // Supports the trap simulation.
    UINT    I;

    // Iterate through the variable bindings list to resolve individual
    // variable bindings.

    for ( I=0; I < variableBindings->len; I++ )
    {
        *errorStatus = ResolveVarBind( &variableBindings->list[I],
                                       requestType );

        // Test and handle case where Get Next past end of MIB view supported
        // by this Extension Agent occurs. Special processing is required to
        // communicate this situation to the Extendible Agent so it can take
        // appropriate action, possibly querying other Extension Agents.
```

Microsoft Windows/NT SNMP Programmer's Reference

```
if ( *errorStatus == SNMP_ERRORSTATUS_NOSUCHNAME &&
    requestType == MIB_ACTION_GETNEXT )
{
    *errorStatus = SNMP_ERRORSTATUS_NOERROR;

    // Modify variable binding of such variables so the OID points
    // just outside the MIB view supported by this Extension Agent.
    // The Extendible Agent tests for this, and takes appropriate
    // action.

    SNMP_oidfree( &variableBindings->list[I].name );
    SNMP_oidcpy( &variableBindings->list[I].name, &MIB_OidPrefix );
    variableBindings->list[I].name.ids[MIB_PREFIX_LEN-1] ++;
}

// If an error was indicated, communicate error status and error
// index to the Extendible Agent. The Extendible Agent will ensure
// that the original variable bindings are returned in the response
// packet.

if ( *errorStatus != SNMP_ERRORSTATUS_NOERROR )
{
    *errorIndex = I+1;
    goto Exit;
}

Exit:

// Supports the trap simulation.

if (++requestCount % 3 == 0 && hSimulateTrap != NULL)
    SetEvent(hSimulateTrap);

// Indicate that Extension Agent processing was successful.

return SNMPAPI_NOERROR;

} // end SmpExtensionQuery()

//----- END -----
```

5.1.3. TESTMIB.C

/*++ BUILD Version: 0001 // Increment this if a change has global effects

Copyright (c) 1991 Microsoft Corporation

Module Name:

testmib.c

Abstract:

Sample SNMP Extension Agent for Windows NT.

These files (testdll.c, testmib.c, and testmib.h) provide an example of how to structure an Extension Agent DLL which works in conjunction with the SNMP Extendible Agent for Windows NT.

Extensive comments have been included to describe its structure and operation. See also "Microsoft Windows/NT SNMP Programmer's Reference".

Created:

13-Jun-1991

Revision History:

--*/

static char *vcSID = "@(#) \$Logfile: N:/xtest/vcs/testmib.c_v \$ \$Revision: 1.2 \$";

```
// This Extension Agent implements the Internet toaster MIB. It's
// definition follows here:
//
//
//      TOASTER-MIB DEFINITIONS ::= BEGIN
//
//      IMPORTS
//          enterprises
//              FROM RFC1155-SMI
//          OBJECT-TYPE
//              FROM RFC-1212
//          DisplayString
//              FROM RFC-1213;
//
//      epilogue      OBJECT IDENTIFIER ::= { enterprises 12 }
```

Microsoft Windows/NT SNMP Programmer's Reference

```
//      toaster      OBJECT IDENTIFIER ::= { epilogue 2 }
//
//      -- toaster MIB
//
//      toasterManufacturer OBJECT-TYPE
//          SYNTAX  DisplayString
//          ACCESS  read-only
//          STATUS  mandatory
//          DESCRIPTION
//              "The name of the toaster's manufacturer. For instance,
//              Sunbeam."
//          ::= { toaster 1 }
//
//      toasterModelNumber OBJECT-TYPE
//          SYNTAX  DisplayString
//          ACCESS  read-only
//          STATUS  mandatory
//          DESCRIPTION
//              "The name of the toaster's model. For instance,
//              Radiant Automatic."
//          ::= { toaster 2 }
//
//      toasterControl OBJECT-TYPE
//          SYNTAX  INTEGER {
//              up(1),
//              down(2)
//          }
//          ACCESS  read-write
//          STATUS  mandatory
//          DESCRIPTION
//              "This variable controls the current state of the
//              toaster. To begin toasting, set it to down(2). To
//              abort toasting (perhaps in the event of an
//              emergency), set it to up(2)."
//          ::= { toaster 3 }
//
//      toasterDoneness OBJECT-TYPE
//          SYNTAX  INTEGER (1..10)
//          ACCESS  read-write
//          STATUS  mandatory
//          DESCRIPTION
//              "This variable controls how well done ensuing toast
//              should be on a scale of 1 to 10. Toast made at 10
//              is generally considered unfit for human consumption;
//              toast made at 1 is lightly warmed."
//          ::= { toaster 4 }
//
//      toasterToastType OBJECT-TYPE
```


Microsoft Windows/NT SNMP Programmer's Reference

```
//          SYNTAX  INTEGER  {
//
//              white-bread(1),
//              wheat-bread(2),
//              wonder-bread(3),
//              frozen-waffle(4),
//              frozen-bagel(5),
//              hash-brown(6),
//              other(7)
//          }
//          ACCESS  read-write
//          STATUS  mandatory
//          DESCRIPTION
//
//              "This variable informs the toaster of the type of
//              material being toasted. The toaster uses this
//              information combined with toasterToastDoneness to
//              compute how long the material must be toasted for
//              to achieve the desired doneness."
//          ::= { toaster 5 }
//
//          END

// Necessary includes.

#include <windows.h>
#include <malloc.h>

#include <snmp.h>

// Contains definitions for the table structure describing the MIB.  This
// is used in conjunction with testmib.c where the MIB requests are resolved.

#include "testmib.h"

// If an addition or deletion to the MIB is necessary, there are several
// places in the code that must be checked and possibly changed.
//
// The last field in each MIB entry is used to point to the NEXT
// leaf variable.  If an addition or deletion is made, these pointers
// may need to be updated to reflect the modification.

// The prefix to all of these MIB variables is 1.3.6.1.4.1.12

UINT OID_Prefix[] = { 1, 3, 6, 1, 4, 1, 12 };
AsnObjectIdentifier MIB_OidPrefix = { OID_SIZEOF(OID_Prefix), OID_Prefix };
```

```
//
//
// OID definitions for MIB //
//
//

// Definition of the toaster group

UINT MIB_toaster[] = { 2 };

// Definition of leaf variables under the toaster group
// All leaf variables have a zero appended to their OID to indicate
// that it is the only instance of this variable and it exists.

UINT MIB_toasterManufacturer[] = { 2, 1, 0 };
UINT MIB_toasterModelNumber[] = { 2, 2, 0 };
UINT MIB_toasterControl[] = { 2, 3, 0 };
UINT MIB_toasterDoneness[] = { 2, 4, 0 };
UINT MIB_toasterToastType[] = { 2, 5, 0 };

//
//
// Storage definitions for MIB //
//
//

char MIB_toasterManStor[] = "Microsoft Corporation";
char MIB_toasterModelStor[] =
    "Example SNMP Extension Agent for Windows/NT (TOASTER-MIB).";
AsnInteger MIB_toasterControlStor = 1;
AsnInteger MIB_toasterDonenessStor = 2;
AsnInteger MIB_toasterToastTypeStor = 3;

// MIB definition

MIB_ENTRY Mib[] = {
    { { OID_SIZEOF(MIB_toasterManufacturer), MIB_toasterManufacturer },
      &MIB_toasterManStor, ASN_RFC1213_DISPSTRING,
      MIB_ACCESS_READ, MIB_leaf_func, &Mib[1] },

    { { OID_SIZEOF(MIB_toasterModelNumber), MIB_toasterModelNumber },
      &MIB_toasterModelStor, ASN_RFC1213_DISPSTRING,
      MIB_ACCESS_READ, MIB_leaf_func, &Mib[2] },
```

```

    { { OID_SIZEOF(MIB_toasterControl), MIB_toasterControl },
      &MIB_toasterControlStor, ASN_INTEGER,
      MIB_ACCESS_READWRITE, MIB_control_func, &Mib[3] },

    { { OID_SIZEOF(MIB_toasterDoneness), MIB_toasterDoneness },
      &MIB_toasterDonenessStor, ASN_INTEGER,
      MIB_ACCESS_READWRITE, MIB_doneness_func, &Mib[4] },

    { { OID_SIZEOF(MIB_toasterToastType), MIB_toasterToastType },
      &MIB_toasterToastTypeStor, ASN_INTEGER,
      MIB_ACCESS_READWRITE, MIB_toasttype_func, NULL }
  };

UINT MIB_num_variables = sizeof Mib / sizeof( MIB_ENTRY );


//
// ResolveVarBind
//   Resolves a single variable binding.  Modifies the variable on a GET
//   or a GET-NEXT.
//
// Notes:
//
// Return Codes:
//   Standard PDU error codes.
//
// Error Codes:
//   None.
//
UINT ResolveVarBind(
    IN OUT RFC1157VarBind *VarBind, // Variable Binding to resolve
    IN UINT PduAction                // Action specified in PDU
)

{
    MIB_ENTRY      *MibPtr;
    AsnObjectIdentifier TempOid;
    int             CompResult;
    UINT            I;
    UINT            nResult;

    // Search for var bind name in the MIB
    I = 0;
    MibPtr = NULL;
    while ( MibPtr == NULL && I < MIB_num_variables )

```

```

{
// Construct OID with complete prefix for comparison purposes
SNMP_oidcpy( &TempOid, &MIB_OidPrefix );
SNMP_oidappend( &TempOid, &Mib[I].Oid );

// Check for OID in MIB - On a GET-NEXT the OID does not have to exactly
// match a variable in the MIB, it must only fall under the MIB root.
CompResult = SNMP_oidcmp( &VarBind->name, &TempOid );
if ( 0 > CompResult )
{
// Since there is not an exact match, the only valid action is GET-NEXT
if ( MIB_ACTION_GETNEXT != PduAction )
{
nResult = SNMP_ERRORSTATUS_NOSUCHNAME;
goto Exit;
}

// Since the match was not exact, but var bind name is within MIB,
// we are at the NEXT MIB variable down from the one specified.
PduAction = MIB_ACTION_GET;
MibPtr = &Mib[I];

// Replace var bind name with new name
SNMP_oidfree( &VarBind->name );
SNMP_oidcpy( &VarBind->name, &MIB_OidPrefix );
SNMP_oidappend( &VarBind->name, &MibPtr->Oid );
}
else
{
// An exact match was found.
if ( 0 == CompResult )
{
MibPtr = &Mib[I];
}
}

// Free OID memory before checking another variable
SNMP_oidfree( &TempOid );

I++;
} // while

// If OID not within scope of MIB, then no such name
if ( MibPtr == NULL )
{
nResult = SNMP_ERRORSTATUS_NOSUCHNAME;
goto Exit;
}

```

```

// Call function to process request.  Each MIB entry has a function pointer
// that knows how to process its MIB variable.
nResult = (*MibPtr->MibFunc)( PduAction, MibPtr, VarBind );

// Free temp memory
SNMP_oidfree( &TempOid );

Exit:
    return nResult;
} // ResolveVarBind

```

```

//
// MIB_leaf_func
//   Performs generic actions on LEAF variables in the MIB.
//
// Notes:
//
// Return Codes:
//   Standard PDU error codes.
//
// Error Codes:
//   None.
//
UINT MIB_leaf_func(
    IN UINT Action,
    IN MIB_ENTRY *MibPtr,
    IN RFC1157VarBind *VarBind
)

{
    UINT ErrStat;

    switch ( Action )
    {
        case MIB_ACTION_GETNEXT:
            // If there is no GET-NEXT pointer, this is the end of this MIB
            if ( MibPtr->MibNext == NULL )
            {
                ErrStat = SNMP_ERRORSTATUS_NOSUCHNAME;
                goto Exit;
            }

            // Setup var bind name of NEXT MIB variable
            SNMP_oidfree( &VarBind->name );
            SNMP_oidcpy( &VarBind->name, &MIB_OidPrefix );

```

```
SNMP_oidappend( &VarBind->name, &MibPtr->MibNext->Oid );
```

```
// Call function to process request. Each MIB entry has a function
// pointer that knows how to process its MIB variable.
```

```
ErrStat = (*MibPtr->MibNext->MibFunc)( MIB_ACTION_GET,
                                       MibPtr->MibNext, VarBind );
```

```
break;
```

```
case MIB_ACTION_GET:
```

```
// Make sure that this variable's ACCESS is GET'able
```

```
if ( MibPtr->Access != MIB_ACCESS_READ &&
    MibPtr->Access != MIB_ACCESS_READWRITE )
{
    ErrStat = SNMP_ERRORSTATUS_NOSUCHNAME;
    goto Exit;
}
```

```
// Setup varbind's return value
```

```
VarBind->value.asnType = MibPtr->Type;
```

```
switch ( VarBind->value.asnType )
```

```
{
case ASN_RFC1155_COUNTER:
case ASN_RFC1155_GAUGE:
case ASN_INTEGER:
    VarBind->value.asnValue.number = *(AsnInteger *) (MibPtr->Storage);
    break;
```

```
case ASN_OCTETSTRING: // This entails ASN_RFC1213_DISPSTRING also
```

```
    VarBind->value.asnValue.string.length =
        strlen( (LPSTR)MibPtr->Storage );
```

```
if ( NULL ==
    (VarBind->value.asnValue.string.stream =
    malloc(VarBind->value.asnValue.string.length *
          sizeof(char))) )
{
    ErrStat = SNMP_ERRORSTATUS_GENERR;
    goto Exit;
}
```

```
memcpy( VarBind->value.asnValue.string.stream,
        (LPSTR)MibPtr->Storage,
```

```
        VarBind->value.asnValue.string.length );
```

```
VarBind->value.asnValue.string.dynamic = TRUE;
```

```
break;
```

```
default:
```

```

        ErrStat = SNMP_ERRORSTATUS_GENERR;
        goto Exit;
    }

    break;

case MIB_ACTION_SET:
    // Make sure that this variable's ACCESS is SET'able
    if ( MibPtr->Access != MIB_ACCESS_READWRITE &&
        MibPtr->Access != MIB_ACCESS_WRITE )
    {
        ErrStat = SNMP_ERRORSTATUS_NOSUCHNAME;
        goto Exit;
    }

    // Check for proper type before setting
    if ( MibPtr->Type != VarBind->value.asnType )
    {
        ErrStat = SNMP_ERRORSTATUS_BADVALUE;
        goto Exit;
    }

    // Save value in MIB
    switch ( VarBind->value.asnType )
    {
        case ASN_RFC1155_COUNTER:
        case ASN_RFC1155_GAUGE:
        case ASN_INTEGER:
            *(AsnInteger *) (MibPtr->Storage) = VarBind->value.asnValue.number;
            break;

        case ASN_OCTETSTRING: // This entails ASN_RFC1213_DISPSTRING also
            // The storage must be adequate to contain the new string
            // including a NULL terminator.
            memcpy( (LPSTR)MibPtr->Storage,
                VarBind->value.asnValue.string.stream,
                VarBind->value.asnValue.string.length );

            ((LPSTR)MibPtr->Storage)[VarBind->value.asnValue.string.length] =
                                                                    '\0';

            break;

        default:
            ErrStat = SNMP_ERRORSTATUS_GENERR;
            goto Exit;
    }

    break;

```

```

default:
    ErrStat = SNMP_ERRORSTATUS_GENERR;
    goto Exit;
} // switch

// Signal no error occurred
ErrStat = SNMP_ERRORSTATUS_NOERROR;

Exit:
    return ErrStat;
} // MIB_leaf_func


//
// MIB_control_func
//     Performs specific actions on the toasterControl MIB variable
//
// Notes:
//
// Return Codes:
//     Standard PDU error codes.
//
// Error Codes:
//     None.
//
UINT MIB_control_func(
    IN UINT Action,
    IN MIB_ENTRY *MibPtr,
    IN RFC1157VarBind *VarBind
)

{
    UINT ErrStat;

    switch ( Action )
    {
        case MIB_ACTION_SET:
            // Make sure that this variable's ACCESS is SET'able
            if ( MibPtr->Access != MIB_ACCESS_READWRITE &&
                MibPtr->Access != MIB_ACCESS_WRITE )
            {
                ErrStat = SNMP_ERRORSTATUS_NOSUCHNAME;
                goto Exit;
            }

            // Check for proper type before setting

```



```

if ( MibPtr->Type != VarBind->value.asnType )
{
    ErrStat = SNMP_ERRORSTATUS_BADVALUE;
    goto Exit;
}

// Make sure the value is valid
if ( MIB_TOASTER_UP > VarBind->value.asnValue.number ||
    MIB_TOASTER_DOWN < VarBind->value.asnValue.number )
{
    ErrStat = SNMP_ERRORSTATUS_BADVALUE;
    goto Exit;
}

// Let fall through purposefully for further processing by
// generic leaf function.

case MIB_ACTION_GETNEXT:
case MIB_ACTION_GET:
    // Call the more generic function to perform the action
    ErrStat = MIB_leaf_func( Action, MibPtr, VarBind );
    break;

default:
    ErrStat = SNMP_ERRORSTATUS_GENERR;
    goto Exit;
} // switch

Exit:
    return ErrStat;
} // MIB_control_func

//
// MIB_doneness_func
//     Performs specific actions on the toasterDoneness MIB variable
//
// Notes:
//
// Return Codes:
//     Standard PDU error codes.
//
// Error Codes:
//     None.
//
UINT MIB_doneness_func(
    IN UINT Action,

```

```

    IN MIB_ENTRY *MibPtr,
    IN RFC1157VarBind *VarBind
)

{
    UINT    ErrStat;

    switch ( Action )
    {
    case MIB_ACTION_SET:
        // Make sure that this variable's ACCESS is SET'able
        if ( MibPtr->Access != MIB_ACCESS_READWRITE &&
            MibPtr->Access != MIB_ACCESS_WRITE )
        {
            ErrStat = SNMP_ERRORSTATUS_NOSUCHNAME;
            goto Exit;
        }

        // Check for proper type before setting
        if ( MibPtr->Type != VarBind->value.asnType )
        {
            ErrStat = SNMP_ERRORSTATUS_BADVALUE;
            goto Exit;
        }

        // Make sure the value is valid
        if ( MIB_TOASTER_LIGHTLYWARM > VarBind->value.asnValue.number ||
            MIB_TOASTER_BURNT < VarBind->value.asnValue.number )
        {
            ErrStat = SNMP_ERRORSTATUS_BADVALUE;
            goto Exit;
        }

        // Let fall through purposefully for further processing by
        // generic leaf function.

    case MIB_ACTION_GETNEXT:
    case MIB_ACTION_GET:
        // Call the more generic function to perform the action
        ErrStat = MIB_leaf_func( Action, MibPtr, VarBind );
        break;

    default:
        ErrStat = SNMP_ERRORSTATUS_GENERR;
        goto Exit;
    } // switch

Exit:

```

```

    return ErrStat;
} // MIB_doneness_func


//
// MIB_toasttype_func
//   Performs specific actions on the toasterToastType MIB variable
//
// Notes:
//
// Return Codes:
//   Standard PDU error codes.
//
// Error Codes:
//   None.
//
UINT MIB_toasttype_func(
    IN UINT Action,
    IN MIB_ENTRY *MibPtr,
    IN RFC1157VarBind *VarBind
)

{
    UINT ErrStat;

    switch ( Action )
    {
    case MIB_ACTION_SET:
        // Make sure that this variable's ACCESS is SET'able
        if ( MibPtr->Access != MIB_ACCESS_READWRITE &&
            MibPtr->Access != MIB_ACCESS_WRITE )
        {
            ErrStat = SNMP_ERRORSTATUS_NOSUCHNAME;
            goto Exit;
        }

        // Check for proper type before setting
        if ( MibPtr->Type != VarBind->value.asnType )
        {
            ErrStat = SNMP_ERRORSTATUS_BADVALUE;
            goto Exit;
        }

        // Make sure the value is valid
        if ( MIB_TOASTER_WHITEBREAD > VarBind->value.asnValue.number ||
            MIB_TOASTER_OTHERBREAD < VarBind->value.asnValue.number )
        {

```

```
        ErrStat = SNMP_ERRORSTATUS_BADVALUE;
        goto Exit;
    }

    // Let fall through purposefully for further processing by
    // generic leaf function.

case MIB_ACTION_GETNEXT:
case MIB_ACTION_GET:
    // Call the more generic function to perform the action
    ErrStat = MIB_leaf_func( Action, MibPtr, VarBind );
    break;

default:
    ErrStat = SNMP_ERRORSTATUS_GENERR;
    goto Exit;
} // switch

Exit:
    return ErrStat;
} // MIB_toasttype_func
```

5.1.4. TESTMIB.H

/*++ BUILD Version: 0001 // Increment this if a change has global effects

Copyright (c) 1991 Microsoft Corporation

Module Name:

testmib.h

Abstract:

Sample SNMP Extension Agent for Windows NT.

These files (testdll.c, testmib.c, and testmib.h) provide an example of how to structure an Extension Agent DLL which works in conjunction with the SNMP Extendible Agent for Windows NT.

Extensive comments have been included to describe its structure and operation. See also "Microsoft Windows/NT SNMP Programmer's Reference".

Created:

13-Jun-1991

Revision History:

```
--*/

#ifndef testmib_h
#define testmib_h

static char *testmib__h = "@(#) $Logfile:  N:/xtest/vcs/testmib.h_v  $ $Revision:  1.2  $";

// Necessary includes.

#include <snmp.h>

// MIB Specifics.

#define MIB_PREFIX_LEN          MIB_OidPrefix.idLength
#define MAX_STRING_LEN          255

// Ranges and limits for specific MIB variables.

#define MIB_TOASTER_UP          1
#define MIB_TOASTER_DOWN        2

#define MIB_TOASTER_LIGHTLYWARM  1
#define MIB_TOASTER_BURNT        10

#define MIB_TOASTER_WHITEBREAD   1
#define MIB_TOASTER_OTHERBREAD   7

// MIB function actions.

#define MIB_ACTION_GET           ASN_RFC1157_GETREQUEST
#define MIB_ACTION_SET           ASN_RFC1157_SETREQUEST
#define MIB_ACTION_GETNEXT       ASN_RFC1157_GETNEXTREQUEST

// MIB Variable access privileges.

#define MIB_ACCESS_READ          0
#define MIB_ACCESS_WRITE         1
#define MIB_ACCESS_READWRITE     2

// Macro to determine number of sub-oid's in array.
```

Microsoft Windows/NT SNMP Programmer's Reference

```
#define OID_SIZEOF( Oid )      ( sizeof Oid / sizeof(UINT) )

// MIB variable ENTRY definition.  This structure defines the format for
// each entry in the MIB.

typedef struct mib_entry
{
    AsnObjectIdentifier Oid;
    void *              Storage;
    BYTE               Type;
    UINT               Access;
    UINT               (*MibFunc)( UINT, struct mib_entry *,
                                   RFC1157VarBind * );
    struct mib_entry *  MibNext;
} MIB_ENTRY;

// Internal MIB structure.

extern MIB_ENTRY Mib[];
extern UINT      MIB_num_variables;

// Prefix to every variable in the MIB.

extern AsnObjectIdentifier MIB_OidPrefix;

// Function Prototypes.

UINT ResolveVarBind(
    IN OUT RFC1157VarBind *VarBind, // Variable Binding to resolve
    IN UINT PduAction              // Action specified in PDU
);

UINT MIB_leaf_func(
    IN UINT Action,
    IN MIB_ENTRY *MibPtr,
    IN RFC1157VarBind *VarBind
);

UINT MIB_control_func(
    IN UINT Action,
    IN MIB_ENTRY *MibPtr,
    IN RFC1157VarBind *VarBind
);
```

```
UINT MIB_doneness_func(  
    IN UINT Action,  
    IN MIB_ENTRY *MibPtr,  
    IN RFC1157VarBind *VarBind  
);
```

```
UINT MIB_toasttype_func(  
    IN UINT Action,  
    IN MIB_ENTRY *MibPtr,  
    IN RFC1157VarBind *VarBind  
);
```

```
#endif /* testmib_h */
```

5.2. Example Manager Application

An example Manager Application Agent has been developed to illustrate the functionality of Win32 Extension Agents. This example agent implements the Toaster MIB that has been used as a simple example to illustrate SNMP.

5.2.1. SNMPUTIL.C

```
/*++ BUILD Version: 0001    // Increment this if a change has global effects
```

Copyright (c) 1991 Microsoft Corporation

Module Name:

snmputil.c

Abstract:

Sample SNMP Management API usage for Windows NT.

This file is an example of how to code management applications using the SNMP Management API for Windows NT. It is similar in operation to the other commonly available SNMP command line utilities.

Extensive comments have been included to describe its structure and operation. See also "Microsoft Windows/NT SNMP Programmer's Reference".

Created:

28-Jun-1991

Revision History:

--*/

```
static char *vcsid = "@(#) $Logfile: N:/agent/mgmtapi/vcs/snmputil.c_v $ $Revision: 1.4 $";
```

```
// General notes:
```

```
// Microsoft's SNMP Management API for Windows NT is implemented as a DLL  
// that is linked with the developer's code. These APIs (examples follow in  
// this file) allow the developer's code to generate SNMP queries and receive  
// SNMP traps. A simple MIB compiler and related APIs are also available to  
// allow conversions between OBJECT IDENTIFIERS and OBJECT DESCRIPTORS.
```

```
// Necessary includes.
```



```

#include <windows.h>

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#include <snmp.h>
#include <mgmtapi.h>

// Constants used in this example.

#define GET      1
#define GETNEXT 2
#define WALK     3
#define TRAP     4

#define TIMEOUT 500 /* milliseconds */
#define RETRIES 3

// Main program.

INT main(
    IN int  argumentCount,
    IN char *argumentVector[])
{
    INT          operation;
    LPSTR        agent;
    LPSTR        community;
    RFC1157VarBindList variableBindings;
    LPSNMP_MGR_SESSION session;

    INT          timeout = TIMEOUT;
    INT          retries = RETRIES;

    BYTE         requestType;
    AsnInteger requestId;
    AsnInteger errorStatus;
    AsnInteger errorIndex;

    // Parse command line arguments to determine requested operation.

    // Verify number of arguments...
    if (argumentCount < 5 && argumentCount != 2)
    {
        printf("Error:  Incorrect number of arguments specified.\n");
    }

```

```

        printf(
"\nusage:  snmputil [get|getnext|walk] agent community oid [oid ...]\n");
        printf(
"        snmputil trap\n");

        return 1;
    }

// Get/verify operation...
argumentVector++;
argumentCount--;
if      (!strcmp(*argumentVector, "get"))
    operation = GET;
else if (!strcmp(*argumentVector, "getnext"))
    operation = GETNEXT;
else if (!strcmp(*argumentVector, "walk"))
    operation = WALK;
else if (!strcmp(*argumentVector, "trap"))
    operation = TRAP;
else
{
    printf("Error:  Invalid operation, '%s', specified.\n",
          *argumentVector);

    return 1;
}

if (operation != TRAP)
{
    if (argumentCount < 4)
    {
        printf("Error:  Incorrect number of arguments specified.\n");
        printf(
"\nusage:  snmputil [get|getnext|walk] agent community oid [oid ...]\n");
        printf(
"        snmputil trap\n");

        return 1;
    }

// Get agent address...
argumentVector++;
argumentCount--;
agent = (LPSTR)malloc(strlen(*argumentVector) + 1);
strcpy(agent, *argumentVector);

// Get agent community...
argumentVector++;

```

```

argumentCount--;
community = (LPSTR)malloc(strlen(*argumentVector) + 1);
strcpy(community, *argumentVector);

// Get oid's...
variableBindings.list = NULL;
variableBindings.len = 0;

while(--argumentCount)
{
    AsnObjectIdentifier reqObject;

    argumentVector++;

    // Convert the string representation to an internal representation.
    if (!SnmMgrStrToOid(*argumentVector, &reqObject))
    {
        printf("Error: Invalid oid, %s, specified.\n", *argumentVector);

        return 1;
    }
    else
    {
        // Since successful, add to the variable bindings list.
        variableBindings.len++;
        if ((variableBindings.list = (RFC1157VarBind *)realloc(
            variableBindings.list, sizeof(RFC1157VarBind) *
            variableBindings.len)) == NULL)
        {
            printf("Error: Error allocating oid, %s.\n",
                *argumentVector);

            return 1;
        }

        variableBindings.list[variableBindings.len - 1].name =
            reqObject; // NOTE! structure copy
        variableBindings.list[variableBindings.len - 1].value.asnType =
            ASN_NULL;
    }
} // end while()

// Make sure only one variable binding was specified if operation
// is WALK.
if (operation == WALK && variableBindings.len != 1)
{
    printf("Error: Multiple oids specified for WALK.\n");
}

```

```
    return 1;
}

// Establish a SNMP session to communicate with the remote agent. The
// community, communications timeout, and communications retry count
// for the session are also required.

if ((session = SnmpMgrOpen(agent, community, timeout, retries)) == NULL)
{
    printf("error on SnmpMgrOpen %d\n", GetLastError());

    return 1;
}

} // end if(TRAP)

// Determine and perform the requested operation.

if (operation == GET || operation == GETNEXT)
{
    // Get and GetNext are relatively simple operations to perform.
    // Simply initiate the request and process the result and/or
    // possible error conditions.

    if (operation == GET)
        requestType = ASN_RFC1157_GETREQUEST;
    else
        requestType = ASN_RFC1157_GETNEXTREQUEST;

    // Request that the API carry out the desired operation.

    if (!SnmpMgrRequest(session, requestType, &variableBindings,
                        &errorStatus, &errorIndex))
    {
        // The API is indicating an error.

        printf("error on SnmpMgrRequest %d\n", GetLastError());
    }
    else
    {
        // The API succeeded, errors may be indicated from the remote
        // agent.

        if (errorStatus > 0)
```

```

    {
        printf("Error: errorStatus=%d, errorIndex=%d\n",
            errorStatus, errorIndex);
    }
else
    {
        // Display the resulting variable bindings.

        UINT i;
        char *string = NULL;

        for(i=0; i < variableBindings.len; i++)
        {
            SnmpMgrOidToStr(&variableBindings.list[i].name, &string);
            printf("Variable = %s\n", string);
            if (string) free(string);

            printf("Value    = ");
            SnmpUtilPrintAsnAny(&variableBindings.list[i].value);

            printf("\n");
        } // end for()
    }
}

// Free the variable bindings that have been allocated.

SnmpUtilVarBindListFree(&variableBindings);

}
else if (operation == WALK)
{
    // Walk is a common term used to indicate that all MIB variables
    // under a given OID are to be traversed and displayed. This is
    // a more complex operation requiring tests and looping in addition
    // to the steps for get/getnext above.

    AsnObjectIdentifier root;
    AsnObjectIdentifier tempOid;

    SnmpUtilOidCpy(&root, &variableBindings.list[0].name);

    requestType = ASN_RFC1157_GETNEXTREQUEST;

```

```

while(1)
{
    if (!SnmprRequest(session, requestType, &variableBindings,
        &errorStatus, &errorIndex))
    {
        // The API is indicating an error.

        printf("error on SnmpMgrRequest %d\n", GetLastError());

        break;
    }
else
    {
        // The API succeeded, errors may be indicated from the remote
        // agent.

        // Test for end of subtree or end of MIB.

        if (errorStatus == SNMP_ERRORSTATUS_NOSUCHNAME ||
            SnmpUtilOidNCmp(&variableBindings.list[0].name,
                &root, root.idLength))
        {
            printf("End of MIB subtree.\n\n");

            break;
        }

        // Test for general error conditions or success.

        if (errorStatus > 0)
        {
            printf("Error: errorStatus=%d, errorIndex=%d\n",
                errorStatus, errorIndex);

            break;
        }
else
        {
            // Display resulting variable binding for this iteration.

            char *string = NULL;

            SnmpMgrOidToStr(&variableBindings.list[0].name, &string);
            printf("Variable = %s\n", string);
            if (string) free(string);
        }
    }
}

```

```

        printf("Value    = ");
        SnmpUtilPrintAsnAny(&variableBindings.list[0].value);

        printf("\n");
    }
} // end if()

// Prepare for the next iteration.  Make sure returned oid is
// preserved and the returned value is freed.

SnmpUtilOidCpy(&tempOid, &variableBindings.list[0].name);

SnmpUtilVarBindFree(&variableBindings.list[0]);

SnmpUtilOidCpy(&variableBindings.list[0].name, &tempOid);
variableBindings.list[0].value.asnType = ASN_NULL;

SnmpUtilOidFree(&tempOid);

} // end while()

// Free the variable bindings that have been allocated.

SnmpUtilVarBindListFree(&variableBindings);

SnmpUtilOidFree(&root);

}
else if (operation == TRAP)
{
    // Trap handling can be done two different ways: event driven or
    // polled.  The following code illustrates the steps to use event
    // driven trap reception in a management application.

    HANDLE hNewTraps = NULL;

    if (!SnmpMgrTrapListen(&hNewTraps))
    {
        printf("error on SnmpMgrTrapListen %d\n", GetLastError());
    }
    else
    {

```

Microsoft Windows/NT SNMP Programmer's Reference

```
printf("snmputil: listening for traps...\n");
}

while(1)
{
    DWORD dwResult;

    if ((dwResult = WaitForSingleObject(hNewTraps, 0xffffffff))
        == 0xffffffff)
    {
        printf("error on WaitForSingleObject %d\n",
            GetLastError());
    }
    else if (!ResetEvent(hNewTraps))
    {
        printf("error on ResetEvent %d\n", GetLastError());
    }
    else
    {
        AsnObjectIdentifier enterprise;
        AsnInteger          genericTrap;
        AsnInteger          specificTrap;
        AsnTimeticks        timeStamp;
        RFC1157VarBindList  variableBindings;

        while(SnmpMgrGetTrap(&enterprise, &genericTrap, &specificTrap,
            &timeStamp, &variableBindings))
        {
            printf("snmputil: trap generic=%d specific=%d\n",
                genericTrap, specificTrap);

            SnmpUtilOidFree(&enterprise);

            SnmpUtilVarBindListFree(&variableBindings);
        }
    }
} // end while()

} // end if(operation)

if (operation != TRAP)
{
    // Close SNMP session with the remote agent.
```


Microsoft Windows/NT SNMP Programmer's Reference

```
if (!SnmprClose(session))
{
    printf("error on SnmpMgrClose %d\n", GetLastError());

    return 1;
}

// Let the command interpreter know things went ok.

return 0;

} // end main()
```

6. References

- [1] RFC 1155 Rose, M.T.; McCloghrie, K. Structure and identification of management information for TCP/IP-based internets. 1990 May; 22 p. (Format: TXT=40927 bytes) (Obsoletes RFC 1065)
- [2] RFC 1157 Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, C. Simple Network Management Protocol (SNMP). 1990 May; 36 p. (Format: TXT=74894 bytes) (Obsoletes RFC 1098)
- [3] RFC 1156 McCloghrie, K.; Rose, M.T. Management Information Base for network management of TCP/IP-based internets. 1990 May; 91 p. (Format: TXT=138781 bytes) (Obsoletes RFC 1066)
- [4] "LAN Manager 2.0 Management Information Base (LanMgr-Mib-II)", Microsoft, May 30, 1991.
- [5] "LAN Manager 2.0 Management Information Base-Alerts (LanMgr-Alerts-II)", Microsoft, April 18, 1991.
- [6] "Microsoft Windows Windows 32-Bit API Programming Reference", Volumes 1&2, Microsoft (Confidential), 1991.
- [7] "Microsoft Windows 32-Bit Development Kit Guide to Programming", Microsoft (Confidential), 1991.
- [8] "Implementation of an SNMP Agent for Microsoft's NT Kernel", Microsoft (Confidential), March 23, 1992.

7. Revision History

July 22, 1992: Delivered Version.

June 28, 1992: Initial Preliminary Version.