

NAME

snmp – Send SNMP messages to devices on the Internet.

DESCRIPTION

The **Tnm** Tcl extension includes an implementation of the Simple Network Management Protocol (SNMP) which is used to monitor and control devices on the Internet. The **Tnm** extension supports SNMP version 1 (SNMPv1) as defined in RFC 1155 and RFC 1157, community based SNMP version 2 (SNMPv2C) as defined in RFC 1901-1908 as well as User-based SNMP version 2 (SNMPv2U) as defined in RFC1909-1910. The Tcl commands described in this man page follow the conventions defined in the SNMPv2 standard (RFC1902-1908). The differences between SNMPv1 and SNMPv2 are hidden as far as possible from the user of this extension by applying automatic conversions where possible.

All SNMP messages are send or received by using so called SNMP sessions. SNMP sessions are light-weight objects that keep control about the transport address of the SNMP peer entity, the authentication mechanism in use as well as some parameters that control the behaviour of the SNMP protocol engine itself.

The programming interface described in this man page can be split in four groups:

- [1] Commands to create and control SNMP sessions.
- [2] Commands to invoke simple SNMP operations on SNMP sessions.
- [3] Commands to invoke complex SNMP operations on SNMP sessions.
- [4] Commands to implement SNMP agents.

The SNMP commands described in this man page are usually used together with the `mib(n)` command which allows to retrieve information from Management Information Base (MIB) definitions.

SNMP DATA TYPES

The SNMP protocol uses a fixed set of primitive data types. These data types are mapped to a primitive string representation in Tcl. Below is the list of SNMP data types with a short description of the corresponding primitive representation in Tcl:

OCTET STRING

An OCTET STRING is a sequence of octect (= bytes). The primitive Tcl string representation for an OCTET STRING value is a string containing hexadecimal numbers (from 00 to ff) separated by colons. For example, the ASCII string "rose" is represented as "72:6f:73:65".

OBJECT IDENTIFIER

OBJECT IDENTIFIER are used to address information in a MIB. An OBJECT IDENTIFIER is a sequence of positive numbers which defines a path through from the root of the global registration tree to a MIB object or MIB instance. The primitive Tcl string representation is the dotted notation, where sub-identifier are separated by dots. For example, the OBJECT IDENTIFIER for the MIB node labelled mib-2 has the Tcl string representation "1.3.6.1.2.1". The **Tnm** extension always returns sub-identifier as decimal numbers. However, the **Tnm** extension accepts sub-identifier which contain hexadecimal values. A hexadecimal sub-identifier is either preceeded by a colon instead of a dot or by the sequence "0x". For example, the OBJECT IDENTIFIER value "1.3.6.1.4:1.0x627" is accepted and converted to the value "1.3.6.1.4.1.1575".

IpAddress

Values of type IpAddress are converted into the usual dotted notation where each byte is converted into a decimal value, which is separated from its predecessor by

a dot. A typical example is "134.169.34.15".

TimeTicks

TimeTicks values are used to represent time intervals. The resolution of a TimeTicks values is a deci-second. TimeTicks values are converted to a string of the form "nd hh:mm:ss.cc" where n is the number of days, hh is the number hours, mm is the number of minutes, ss is the number of seconds and cc is the number of hundreds of a seconds. A typical primitive Tcl string representation of a TimeTicks value looks like "48d 12:36:43.22". The **Tnm** extension accepts TimeTicks values where the number of days and/or the number of deci-seconds is missing, e.g. "1164:36:43". The deci-seconds are set to 0 if they are missing.

INTEGER, Integer32, Counter, Counter32, Gauge, Gauge32, Unsigned32

Values of all these numeric types are converted into an integer number. Counter, Counter32, Gauge, Gauge32 and Unsigned32 values are always positive. INTEGER or Integer32 values can be negative. The Counter32, Gauge32 and Integer32 types are limited to 32 bits.

Counter64

The Counter64 type allows to use 64 bit unsigned counters. A value of the Counter64 type is mapped to an integer number on 64 bit machines and to a floating point number on 32 bit machines.

The **Tnm** extension returns values in the primitive string representation if no other formatting rules apply. Other formatting rules are extracted from MIB specifications (see the `mib(n)` command on how to load MIB specifications) and applied automatically. This includes the interpretation of DISPLAY-HINTs as well as the conversion of integer values to the corresponding enumeration. For example, DisplayString values are automatically converted from the primitive OCTET STRING representation into an ASCII string by applying the "225a" DISPLAY-HINT and values of type TruthValue are represented as "true" or "false" instead of 1 and 2 (RFC 1903). Explicit conversions are possible using the **mib format** and **mib scan** commands described in the `mib(n)` man page.

SNMP VARBIND LISTS

The SNMP protocol always operates of a list of MIB instances. This list is called a varbind list. A varbind list is represented as a Tcl list. Each element of the varbind list is itself a Tcl list describing one varbind element. A fully specified varbind list element has three elements: an object identifier which identifies the MIB instance, the base type of the MIB instance and the value of the MIB instance. A fully specified varbind list might look like:

```
{
  {1.3.6.1.2.1.1.3.0 TimeTicks {0d 0:03:23.08}}
  {1.3.6.1.2.1.2.2.1.3.1 INTEGER softwareLoopback}
  {1.3.6.1.2.1.2.2.1.2.1 {OCTET STRING} lo0}
  {1.3.6.1.2.1.2.2.1.8.1 INTEGER up}
}
```

It is often not necessary to use fully specified varbind lists. It is possible to omit the type and value fields if you are retrieving values of MIB instances. For example, to retrieve the fully specified varbind list example above, the following much shorter varbind list has been used:

```
{
  1.3.6.1.2.1.1.3.0
  1.3.6.1.2.1.2.2.1.3.1
  1.3.6.1.2.1.2.2.1.2.1
  1.3.6.1.2.1.2.2.1.8.1
}
```

It is also allowed to use object type descriptors instead of object identifier in dotted notation. (See the `mib(n)` man page for details about object identifier names.) This further simplifies the varbind list above to:

```
{ sysUpTime.0 ifType.1 ifDescr.1 ifOperStatus.1 }
```

In order to change a MIB instance, you have to include the new value in the varbind list. It is however possible to omit the type element if the type can be found in a MIB module. This allows to use a list like

```
{ {sysContact.0 "schoenw@cs.utwente.nl"} }
```

to change to value of the `sysContact.0` instance. Please note, that the **Tnm** extension always returns fully specified varbind list to avoid ambiguities.

SNMPv2 varbind list may contain exceptions (`noSuchObject`, `noSuchInstance`, `endOfMibView`). These exceptions are signalled in the type element of a varbind list element. The value of the varbind is set to null value which conforms to the type of the object type:

```
{
  { 1.3.6.1.2.1.2.2.1.3.2 noSuchInstance 0 }
  { 1.3.6.1.2.1.2.2.1.2.2 noSuchInstance {} }
}
```

It is the responsibility of the application to check for exceptions. The application will not necessarily crash if such a check is missing but it might get confused while interpreting null values.

SNMP SESSION OPTIONS

SNMP sessions are configured by manipulating session options. Some of these options are the same for all supported SNMP versions while some of them are specific to a particular version. The options described below can be used with any SNMP session unless stated otherwise.

-address *address*

The **-address** option defines the network address of the SNMP peer. The value of *address* may be an IP address in dotted notation (e.g. 134.169.34.1) or a hostname that can be resolved to an IP address. The default address is 127.0.0.1.

-port *port*

The **-port** option defines the UDP port which is used by the SNMP peer to receive SNMP messages. The value of *port* may be a port number or a service name which can be resolved to a port number. The default port number is 161.

-version *number*

The **-version** option selects the SNMP version used by a SNMP session. The currently supported version numbers are SNMPv1 (RFC 1157 style SNMP), SNMPv2C (RFC 1901 style SNMP) and SNMPv2U (RFC 1910 style SNMP). The default SNMP version is SNMPv1.

-community *string*

The **-community** option is specific for SNMPv1 and SNMPv2C sessions. It defines the community string which is used to identify the sender of SNMP messages. The default community string is "public".

-writecommunity *string*

The **-writecommunity** option is specific for SNMPv1 and SNMPv2C sessions. It defines the community string which is used to identify the sender of SNMP messages when invoking SNMP set operations. The default write community string is empty which means that the community string defined by the **-community** option is used.

-user *name*

The **-user** option is specific to SNMPv2U sessions. It is used to define the SNMP user name. The default user name is "public".

-password *password*

The **-password** option is specific to SNMPv2U sessions. It is used to specify the password of the user. The password is automatically converted into a key by applying the password2key algorithm of RFC 1910. The key is also automatically localized once the agentID of the SNMP agent is known. Note that the application should take care to keep the passwords safe from unauthorized access. An empty password turns SNMPv2U authentication off. The default password is "".

-context *context*

The **-context** option is specific to SNMPv2U sessions. It allows to select between multiple contexts. The default context is "".

-timeout *time*

The **-timeout** option defines the time the session will wait for a response. The *time* is defined in seconds with a default of 5 seconds.

-retries *number*

The **-retries** option defines how many times a request is retransmitted during the timeout interval. The default *number* of retries is 3.

-delay *delay*

The **-delay** option can be used to define a delay in milliseconds between two messages send by the SNMP protocol engine. This can be used to avoid network congestion problems. The default *delay* is 0 milliseconds.

-window *size*

The **-window** option allows to define a window which limits the number of active asynchronous requests. This can be used to prevent fast scripts to flood an agent with asynchronous messages. The **Tnm** extension queues requests internally so that no more than *size* asynchronous requests are on the wire. Setting the size to 0 turns the windowing mechanism off. The default window size is 10 messages.

-agent *interp*

The **-agent** option turns the SNMP session into an SNMP agent. The *interp* argument defines the Tcl interpreter which will be used to evaluate bindings associated with MIB instances. It is recommended to use a safe Tcl interpreter if you are evaluating scripts or arguments received over the network. It is also possible to use the current Tcl interpreter by using an empty interpreter name.

-alias *name*

The **-alias** option substitutes this option with the configuration options contained in the alias identified by *name*. You can define configuration option aliases with the **snmp alias** command. It is possible to refer to other **-alias** options within an **-alias** option. See the description of the **snmp alias** command below for an example.

SNMP CALLBACK SCRIPTS

Many SNMP commands described below allow to program asynchronous SNMP operations. Asynchronous SNMP operations work by sending out a request without waiting for a response. The SNMP protocol engine keeps track of asynchronous requests and processes callback scripts once an answer for an asynchronous request is received or the request times out. The callback script is always evaluated at global level. Special % escape sequences can be used in the callback script to access details contained in the SNMP response. These % escape sequences are substituted before the callback script is evaluated. The substitution depends on the character following the %, as defined in the list below.

%% Replaced with a single percent.

%V Replaced with the fully specified varbind list.

%R Replaced with the request id.

%S Replaced with the session name.

%E Replaced with the error status. Possible values for the error status are noError, tooBig, noSuchName, badValue, readOnly, genErr, noAccess, wrongType, wrongLength, wrongEncoding, wrongValue, noCreation, inconsistentValue, resourceUnavailable, commitFailed, undoFailed, authorizationError, notWritable, inconsistentName, noResponse.

%I Replaced with the error index. The first element of the varbind list has the position 1.

%A Replaced with the IP address of the peer sending the packet.

%P Replaced with the port number of the peer sending the packet.

%T Replaced with the SNMP PDU type. Possible values for the PDU type are get-request, get-next-request, response, snmpV1-trap, set-request, get-bulk-request, inform-request, snmpV2-trap, report.

SNMP COMMAND

This section describes SNMP commands that are used to create new SNMP sessions, to define SNMP aliases and to do some housekeeping.

snmp session [*option value ...*]

The **snmp session** command creates new SNMP sessions. It returns a session handle which is used to invoke operations. It is possible to add configuration options as described above to the **snmp session** command in order to configure the SNMP session.

snmp alias [*name [options]*]

The **snmp alias** command allows to create aliases for a list of configuration options. This can be used to define short names for complex session configurations and it allows to share e.g. frequently used community strings between a number of devices. An alias defined by the **snmp alias** command can be substituted when configuring a SNMP session by using the **-alias** session option as described above.

The **snmp alias** command invoked without arguments returns the list of all known alias names. If

invoked with an alias *name*, the list of configuration options associated with this alias is returned. Invoking the **snmp alias** command with an alias *name* and an *options* list associates the list of configuration options contained in *options* with the alias *name*.

Below is an examples that creates an SNMPv1 alias private which defines a SNMPv1 community string. The aliases hub1 and hub2 define configuration options used to access two hubs. The aliases hub1/private and hub2/private combine these aliases. It is now possible to update the private community string for both hubs by changing the private alias. Nesting alias definitions as shown below allows to build a flexible configuration scheme.

```
snmp alias private "-community ncc1701"
snmp alias hub1 "-address 1.2.3.4 -window 2 -delay 10"
snmp alias hub2 "-address 1.2.3.5. -timeout 10"
snmp alias hub1/private "-alias hub1 -alias private"
snmp alias hub2/private "-alias hub1 -alias private"
```

snmp info

The **snmp info** command returns a list of existing SNMP sessions.

snmp watch toggle

The **snmp watch** command turns hex printing of SNMP packets on or off. This is mostly a debugging aid because you can't process the output with a Tcl script.

snmp wait

The **snmp wait** command blocks until all asynchronous requests for all SNMP sessions have been processed. Events are processed while waiting for outstanding responses which can have arbitrary side effects.

SIMPLE SNMP SESSION COMMANDS

This sections describes simple SNMP session commands which correspond directly to SNMP protocol operations. It also discusses some commands to configure and control of the SNMP protocol engine.

snmp# configure [*option value ...*]

The **snmp# configure** session command can be used to query or change the current configuration options. See the description of SNMP session options above for more details. The **snmp# configure** session command always returns the current settings of the configuration options. Note, invoking the **snmp# configure** session command on a SNMP session that has asynchronous requests pending can have arbitrary side effects because it processes events until all pending requests for this session have either timed out or returned a response. This behavior makes sure that pending request are not affected by the any change to a configuration option.

snmp# cget *option*

The **snmp# cget** session command allows to retrieve the value of the SNMP session option given by *option*. See the description of SNMP session options above for more details.

snmp# wait [*request*]

The **snmp# wait** command blocks until all asynchronous requests for this SNMP session have been processed. Events are processed while waiting for outstanding responses which can have arbitrary side effects.

snmp# destroy

The **snmp# destroy** session command destroys the session. All data associated with the SNMP session is removed (outstanding asynchronous requests are cancelled).

snmp# get *varbindlist* [*script*]

The **snmp# get** session command retrieves the list of instances as specified in the *varbindlist* by using an SNMP get-request. If the **snmp# get** session command contains a callback *script*, then the command sends the get-request and returns immediately. The result is the request id for the asynchronous request.

The **snmp# get** session command is processed synchronously if the callback *script* is missing. In this case, the received varbind list is returned or the command fails if the agent does not respond or if a protocol error happens.

Below is an example for a synchronous and an asynchronous get request to retrieve variables of the system group:

```
$s get "sysDescr.0 sysName.0 sysContact.0"
```

```
$s get "sysDescr.0 sysName.0 sysContact.0" {
    if {"%E" == "noError"} { puts "%V" }
}
```

snmp# getnext *varbindlist* [*script*]

The **snmp# getnext** session command retrieves the values of the lexicographical successors to the objects named in *varbindlist*. If the **snmp# getnext** session command contains a callback *script*, then the command sends the get-next-request and returns immediately. The result is the request id for the asynchronous request.

The **snmp# getnext** session command is processed synchronously if the callback *script* is missing. In this case, the received varbind list is returned or the command fails if the agent does not respond or if a protocol error happens.

To retrieve the whole MIB tree, you can use the result of a getnext command as an argument for the next getnext command. Below are two examples that show a synchronous and an asynchronous version.

```
set vbl 1
while {![catch {$s getnext $vbl} vbl]} {
    puts $vbl
}

proc dump {s vbl err} {
    if {$err == "noError"} {
        puts $vbl
        $s getnext $vbl { dump "%S" "%V" "%E" }
    }
}
$s getnext 1 { dump "%S" "%V" "%E" }
```

snmp# getbulk *nr mr varbindlist [script]*

The **snmp# getbulk** session command can be used to implement fast MIB tree walks by using get-bulk-requests. The **snmp# getbulk** command performs a getnext on the first *nr* elements given in the *varbindlist*. For the remaining elements, the agent is asked to repeat the getnext operation mostly *mr* times. SNMPv1 sessions automatically map get-bulk-requests to get-next-requests. If the **snmp# getbulk** session command contains a callback *script*, then the command sends the get-bulk-request and returns immediately. The result is the request id for the asynchronous request.

The **snmp# getbulk** session command is processed synchronously if the callback *script* is missing. In this case, the received varbind list is returned or the command fails if the agent does not respond or if a protocol error happens.

A typical example of the **snmp# getbulk** session command is shown below. It will return sysUpTime.0 and the values for ifIndex.1, ifDescr.1, ifIndex.2 and ifDescr.2 assuming the device has at least 2 interfaces and implements snmp version 2.

```
$s getbulk 1 2 "sysUpTime ifIndex ifDescr"
```

snmp# set *varbindlist [callback]*

The **snmp# set** session command can be used to create and alter MIB instances. The varbind list for set-requests must contain at least the object identifier and the new value as described above. If the **snmp# set** session command contains a callback *script*, then the command sends the set-request and returns immediately. The result is the request id for the asynchronous request.

The **snmp# set** session command is processed synchronously if the callback *script* is missing. In this case, the received varbind list is returned or the command fails if the agent does not respond or if a protocol error happens.

Below is an example which shows how you can change the sysContact.0 and sysLocation.0 variables of the system group. Note that it is always recommended to use the Tcl list command to build the varbind list. This makes sure that quoting is done in the correct way.

```
$s set [list \
    [list sysContact.0 "Bert Nase"] \
    [list sysLocation.0 "Hall6, Floor 5"] \
]
```

snmp# trap *snmpTrapOid varbindlist*

The **snmp# trap** command allows to notify a manager that a special event has happened. The trap type is defined by the *snmpTrapOid* object identifier. The standard traps defined in RFC 1907 are coldStart, warmStart and authenticationFailure. Additional trap types are defined in other MIB modules. The *varbindlist* contains variables that provide additional information for the remote manager. Note, the value of sysUpTime.0 is added to the *varbindlist* automatically. However, it is up to the user to provide any other required additional varbinds like the ifIndex for linkUp and linkDown traps.

SNMPv1 traps are defined by an enterprise object identifier and a generic and specific trap number. It is possible to convert a SNMPv1 trap definition into a *snmpTrapOid* object identifier by appending the generic and specific trap number to the enterprise object identifier. This conversion is done automatically if the SNMPv1 trap is defined using a TRAP-TYPE macro (RFC 1212) in a MIB module.

SNMP traps are usually received on UDP port 162. It is therefore necessary to change the default settings of the SNMP session handle to send a trap. Below is an example that changes the destination port, sends a coldStart trap, and restores the destination port. It is also possible to keep a special session around just for sending trap messages.

```
set port [$s cget -port]
$s configure -port 162
$s trap coldStart [list [list sysDescr.0 "Fridge"]]
$s configure -port $port
```

snmp# inform *snmpTrapOid varbindlist [script]*

The **snmp# inform** session command is only available for SNMPv2 sessions and sends an inform-request to a manager. In contrast to a trap, an inform-request is confirmed by the manager with a response message. The information request type is defined by the *snmpTrapOid* in the same way as described for the **snmp# trap** session command above. The *varbindlist* contains variables that provide additional information for the remote manager. Note, the value of sysUpTime.0 is added to the *varbindlist* automatically.

If the **snmp# inform** session command contains a callback *script*, then the command sends the inform-request and returns immediately. The result is the request id for the asynchronous request.

The **snmp# inform** session command is processed synchronously if the callback *script* is missing. In this case, the received varbind list is returned or the command fails if the agent does not respond or if a protocol error happens.

```
set port [$s cget -port]
$s configure -port 162
$s inform tooHot [list [list sysDescr.0 "Fridge"]]
$s configure -port $port
```

snmp# bind {} send *[script]*
snmp# bind {} recv *[script]*
snmp# bind {} trap *[script]*
snmp# bind {} inform *[script]*
snmp# bind {} report *[script]*

The **snmp# bind** session command allows to bind scripts to events processed inside of the SNMP protocol engine. The first argument of the **snmp# bind** session commands described here is always an empty string. (See the description of the agent commands below for additional bindings and the use of this parameter.) The second argument of the **snmp# bind** session command is the event type. The **snmp# bind** session command always returns the script currently bound to the given event type. A script is bound to an event type by providing the optional *script* argument. A script is removed from an event by binding an empty string to this event.

A *send* event is created whenever a SNMP message is sent to the network. Note, that the *send* event only triggers once for every request and not for every retransmission of the same message. Similarly, the *recv* event is created whenever a SNMP message is received from the network. These commands can be used to analyze the SNMP traffic from/to the SNMP engine.

Notifications sent by an agent or another manager can be received by binding scripts to *trap* and *inform* events. Binding a script to one of these two event types starts the straps(8) daemon which listens for incoming notifications on UDP port 162 and forwards them to all interested processes

on the local machine.

A *report* event is created when the protocol engine receives a SNMPv2U report message. These messages are used internally to implement clock synchronization and agent discovery.

A script bound to an event is evaluated in the same way as callback scripts. Substitution of % escape sequence takes place as described above and scripts are always evaluated at global level.

COMPLEX SNMP SESSION COMMANDS

This section describes complex SNMP commands which usually perform a sequence of SNMP operations. The implementation of these complex SNMP session commands does some automatic error handling in order to hide some of the differences between the SNMP version.

snmp# walk *varName varbindlist body*

The **snmp# walk** session command walks a whole MIB subtree. The command repeats sending getbulk requests until the returned varbind list is outside of the subtree given by *varbindlist*. For each valid varbind list retrieved from the agent, the Tcl script *body* is evaluated. Before evaluation of *body* starts, the variable named *varName* is set to contain the actual varbind list. Below is a simple example to walk and print the interface table:

```
$s walk x "ifIndex ifDescr" { puts $x }
```

snmp# scalars *scalarlist arrayName*

The **snmp# scalars** session command can be used to retrieve potentially large lists of scalars. The values are extracted from the varbindlist and stored in the Tcl array named *arrayName*. The *scalarlist* may include names of simple scalar MIB instances or names of SNMP groups which can be expanded to a list of MIB scalars. The array *arrayName* is indexed by the names of scalars successfully retrieved from the agent. The list of indexes is returned by the **snmp# scalars** session command. Protocol errors (e.g. tooBig) or SNMPv2 varbind exceptions are handled internally.

The example below retrieves and prints the system uptime and the all the scalars in the udp and tcp groups:

```
foreach name [$s scalars "sysUpTime udp tcp" s] {
    puts "$name : ${s($name)}"
}
```

SNMP AGENT COMMANDS

This section describes the SNMP session commands used to implement SNMP agents in Tcl. The SNMP agent maintains a tree of MIB instances which are linked to Tcl variables. Whenever the value of a MIB instance is required by the SNMP protocol engine, the corresponding Tcl variable is read. This allows to update MIB instances easily from Tcl. However, some MIB instances are designed to have arbitrary side effects, especially if the value is changed. A binding mechanism allows to bind Tcl scripts to events inside of the SNMP protocol engine so that a Tcl script can control the behavior of MIB instances.

snmp# instance *label varName [default]*

The **snmp# instance** session command creates a new MIB instance if the session is configured as an SNMP agent. The MIB instance identified by *label* is linked to the global Tcl variable *varName*. The SNMP protocol engine reads the value of *varName* and converts it into the required SNMP data type while processing incoming requests. An agent script must ensure that *varName* contains an acceptable format (see above). The optional argument *default* defines the initial value

of *varName*. The lifetime of the MIB instance is bound to the Tcl variable *varName*.

snmp# bind *label event [script]*

The **snmp# bind** session command binds a Tcl *script* to the MIB instance tree node identified by *label* and is only valid for sessions in agent mode. The *event* argument defines the SNMP operation that triggers the evaluation of the *script*. Supported events types are get, set, create, check, commit, rollback, begin and end. The **snmp# bind** session command returns the currently defined binding if the *script* argument is missing.

The begin and end bindings are evaluated before PDU processing starts and after PDU processing has finished. The *label* for begin and end bindings must be empty. Get event bindings are evaluated, whenever a MIB instance is read. They can be used to modify the contents of the associated Tcl variable before the value is actually put into the response message.

Set, create, check, commit and rollback bindings are used to implement writable MIB instances. The check, commit and rollback bindings are needed to allow scripts to conform to the "as if simultaneously" requirement (RFC 1157, RFC 1905). Set requests are processed in three phases. In the first phase, set and create events are processed to modify existing or create new MIB instances. The second phase is processed only if no error occurred in the first phase. The second phase activates check bindings to allow scripts to check the consistency of the new values. The final phase is the commit/rollback phase which triggers the commit bindings if there has not been any error in the earlier phases. An error in one of the earlier phases triggers the rollback bindings. The protocol engine will also restore the Tcl variables with previously saved values.

A Tcl script bound to an event can signal a SNMP specific error by invoking the Tcl error command and returning one of the SNMP error codes (tooBig, noSuchName, badValue, readOnly, genErr, noAccess, wrongType, wrongLength, wrongEncoding, wrongValue, noCreation, inconsistentValue, resourceUnavailable, commitFailed, undoFailed, authorizationError, notWritable, inconsistentName).

All the % escapes sequences described in the section about callback scripts will be expanded before the command is evaluated. In addition, there are three more escapes defined for SNMP bindings:

- %o** Replaced with the object identifier of the MIB instance that triggered the event.
- %i** Replaced with the instance identifier of the MIB instance that triggered the event.
- %v** Replaced with the value for the MIB instance that triggered the event.
- %p** Replaced with the previous value for the MIB instance during SNMP set operations and an empty string otherwise.

Below is an example that implements a MIB instance `tnmTclCmdCount.0` which returns the actual number of Tcl commands processed by the Tcl interpreter:

```
$s instance tnmTclCmdCount.0 tnmTclCmdCount
$s bind tnmTclCmdCount.0 get {
    set tnmTclCmdCount [info cmdcount]
}
```

The second example implements two MIB instances called `tnmHttpSource.0` and `tnmHttpError.0`. The `tnmHttpSource.0` instance shall retrieve a Tcl file via HTTP and evaluate the contents. The second `tnmHttpError.0` instance contains the error message of the last use of `tnmHttpSource.0`:

```

$s instance tnmHttpSource.0 tnmHttpSource
$s instance tnmHttpError.0 tnmHttpError

$s bind tnmHttpSource.0 set {
    set tnmHttpSource "%v"
    set msg [SNMP_HttpSource $tnmHttpSource]
    if {$msg != ""} {
        set tnmHttpError $msg
        error inconsistentValue
    }
}

$s bind tnmHttpSource.0 rollback {
    set tnmHttpSource ""
}

$s bind tnmHttpSource.0 commit {
    set tnmHttpError ""
}

```

It is sometimes useful to bind scripts to non-leaf nodes of the MIB instance tree. All bindings starting from the leaf node up to the root of the MIB instance tree are processed. You can use the Tcl break command to disable further binding processing. For example, the following binding will trigger on all get events in the enterprise MIB subtree and can be used for debugging purposes:

```

$s bind 1.3.6.1.4.1 get {
    puts "%T from %A:%P object %o (instance %i)"
}

```

BUGS

The Tcl arithmetic is not platform independent. It is therefore complicated to write portable scripts that work correctly with large SNMP numbers.

The complex SNMP session commands should support asynchronous operations.

SEE ALSO

scotty(1), Tnm(8), Tcl(n)

AUTHORS

Juergen Schoenwaelder <schoenw@cs.utwente.nl>

Sven Schmidt <vschmidt@ibr.cs.tu-bs.de>