TANDBERG SNMP

TANDBERG

Application Notes

D12190 Rev. 5

Table of Contents

Using TANDBERG SNMP

1. Introduction

The Codec supports the SNMP (Simple Network Management Protocol) standard for network management and surveillance. SNMP is the de facto standard in network management for IP-based units in a network.

This document describes how to use and configure the TANDBERG SNMP implementation.

See also:

D1214800-Tsnmp-Doc.doc (Tandberg SNMP Tool documentation)
D1214900-Tboot-Doc.doc (Tandberg Tboot, bootlogger documentation)

For the level of support in various software releases, see section **10. Software Version Differences.**

2. Acronyms

SNMP	=	**S**imple **N**etwork **M**anagement **P**rotocol
IETF	=	**I**nternet **E**ngineering **T**ask **F**orce
RFC	=	**R**equest **F**or **C**omment
MIB	=	**M**anagement **I**nformation **B**ase

3. SNMP Introduction

SNMP is an IETF standard for management of network based units. SNMP has been in use for a long time, and is the only standardized network management protocol that is widely used today.

For SNMP environments, at least two separate software solutions are in use. Each network unit (for example the TANDBERG codec) has an SNMP *Agent* that handles requests from the controlling software, the SNMP *Manager*. The SNMP Manager will send requests for information or status, or instructions to set specific variables to the SNMP Agent.

The supported variables are logically organized in a tree structure, which is known as the MIB (Management Information Base).The definition of the variables in the MIB tree, known as “MIB files”, “MIB definitions” or similar is either standardized by organizations like the IETF or ITU, or is vendor specific. The “MIB tree” is organized in a manner that there will be no two MIB files with conflicting placement in this global tree.

The IETF MIB definitions are collected in RFCs (Request for Comments), which is the same series of documents that describes nearly every other Internet standard as well. RFCs typically have only a number to identify them, for example RFC1157 is one of the documents describing the SNMP version 1 standard.

SNMP uses the UDP/IP protocol for sending and receiving messages. This means that in a heavily loaded network, certain SNMP packages can get lost. It is up to the SNMP Manager program to implement retransmission of requests if necessary.

4. The SNMP Agent

4.1 Enabling SNMP

SNMP is always enabled in the Codec. The SNMP Agent is started when the Codec is rebooted. If SNMP is not in use, it does not in any way conflict with other networking equipment or the use of the rest of the functionality in the Codec.

The TANDBERG SNMP Agent implements version 1 of SNMP. This means that it does not implement the “get-bulk” operator or enhanced security features of some of the SNMP version 2 implementations and/or SNMP version 3.

4.2 MIB implementation

The software fully supports the mib-2 definitions, as defined in RFC1213. This is the most essential MIB group, which contains system information, networking status, accounting and interface descriptions. Since the Codec is not IP networking hardware some interfaces will have 0 values where an Ethernet type interface is assumed in RFC1213.

In addition an enterprise MIB is defined. TANDBERG’s enterprise MIB has been allocated OID number 1.3.6.1.4.1.**5596**.

4.3 TANDBERG Enterprise MIB

The TANDBERG Enterprise MIB currently holds a lot of information about the status of the Codec. The information are divided into:

- status
- logging
- services
- system

In addition, a few traps are defined to send SNMP Managers information about important events happening on the system. Next is a brief overview of the information in the various parts of the Enterprise MIB.

4.3.1 Status

The status part of the MIB further divided into

- callstatus
- callTable
- incomingTable
- outgoingTable

Callstatus has a few parameters that gives the current state of the codec, such as if it is in a video conference, a multisite set up, a telephone call or similar. In addition an entry giving the last error message, and another showing the “time since restart” the error occurred (as a side note, the mib2 SysTime variable can be used to find out how long the system has been running). In B4.0 the callstatus also contains callLastCauseCode and callLastCauseCodeLoc.

CallNumber inidicates how many entries there are in the CallTable. CallTable is a table containg different status variables with regards to the current call, such as foreign site number, direction of call, how many channels are used and so on. CallID is a number indicating which channel this is, while CallTag is a unique identifier since reboot for this call session. This CallTag is transferred to the logTable (see below) when the call ends, so that unique calls can be identified. In B4.0 the calltable also contains callAccNo and callEncryption.

IncomingTable contains information specific to the incoming call, such as what encoding standards for audio and video that are used and what the resolution for the video are.

OutgoingTable is similar to the IncomingTable except for the fact that this is what this system is sending.

4.3.2 Logging

The logtable contains (for B3.0 software release and further on) a copy of the callTable after a session has ended. Currently 20 calls are being logged. Since this log contains information about what protocols are being used (such as h323, h221, leased line and so on) and the amount of time spent for this call, enough data to be used for billing can be retrieved. The unique callTag (since time of boot) can be used to ensure that a single call is not billed twice. It is up to external application software to use this functionality. The new variables in the callTable is also added here.

4.3.3 Services

The two fields in the services part indicates whether the two services streaming and VNC are enabled. VNC is a presented as a “video source” in the system, while streaming is transparent.

4.3.4 System

The System part holds information about the codec hardware and software. In addition it holds the user configured local isdn phone number for the unit. The software id field is an ever increasing digit for each new software release. In the B4.0 release sytem also

contains information whether protect is on or off (protect), wheter ippass is set or not (ippass) and the 323alias (h323alias).

4.3.5 Traps

For generic information about SNMP traps see section 7 below.

The MIB2 interface traps for “system coldstart”, “interface up” and “interface down” are used. “System warmstart” is not used, since the SNMP agent in the Codec will only be started at boot. In addition the following enterprise traps are defind.

- callModeChange
- genericError
- lowBattery
- downSpeeding
- startUpgrade
- finishedUpgrade
- dispbox
- connect
- encryption
- gatekeeperRegFailure
- gatekeeperRegSuccess

CallModeChange will be sent whenever the codec goes from one kind of status to another. For example, it can change from being “Idle” to “VideoConf” to “MultiSite”. When this trap is received, management software can expect several of the MIB status variables to be changed, at least callMode will be.

GenericError is sent when there is some kind of failure in the system. The exact failure descriptor can be read out from the “lastError” MIB entry, and the exact time since reboot can be read from the “lastErrorTime” entry.

LowBattery is sent out when the remote control is running low on power. Change the batteries in the remote control when this occurs.

DownSpeeding is sent when, for some reason, the system cannot sustain the original requested bandwidth. This can be due to network errors or other parameters.

StartUpgrade will be sent when an upload of new software to the codec has been initiated.

FinishedUpgrade will be sent when an upload of new software to the codec is finished

Dispbox trap is sent when there has been a response from the user (or the system) on a messageBox on the codec.

A connect trap is sent when the call is actually conncected. This is not to be confused with the the callModeChange trap. A callModeChangeTrap does not necessarily mean the call is connected.

A encryption trap is sent when the outgoing encryption-status is encrypted (with the DES algorithm in the B4.0 software).

4.4 Options for configuring

A few of the mib-2 variables can be overwritten by SNMP Management software or by using the dataport or WWW interface. The writable variables corresponds directly with the ones defined in RFC1213. Trying to write over other, read-only variables will result in an error.

5. SNMP Managers

Any SNMP Manager supporting SNMPv1 will interface with the Codec. SNMP Managers can be found from several commercial vendors, and a few “open source” packages also exist. A simple “test” SNMP Manager (TSNMP) has been made by TANDBERG. HP OpenView and IBM NetView are packages using SNMP which can be adopted to large scale networks, and the TANDBERG codecs should integrate nicely with these, although there are no extended management modules for TANDBERG videoconferencing units.

There also exists several open source SNMP implementations that can be used to access the MIB in the Codec. Extensions and libraries for C, Perl, Python and other programming languages exist.

A few references are given here:

Description	**URL**
University of California at Davis, SNMP package with libraries and utilities for C/C++ programmers	http://ucd-snmp.ucdavis.edu
For a list of vendors supporting SNMP (including the newly established SNMPv3 standard)	http://www.ibr.cs.tu-bs.de/ietf/snmpv3
SNMP module for Perl	http://www.switch.ch/misc/leinen/snmp/perl
SNMP module for Python	http://www.glas.net/~ilya/software/pysnmp.html

6. Configuring TANDBERG SNMP

6.1 SNMPv1 Security

SNMP can be used (especially towards networking equipment) to disable networking interfaces and configure a lot of different settings. To protect from “unauthorized” configuration, SNMPv1 has a simple “password” scheme to protect the units from being managed by anybody; the *SNMP Community Name*.

This “SNMP password” is inserted into every SNMP package (request/response) to verify that the request/response is authorized.

Unfortunately this password scheme does not provide very good protection, as the Community Name is transmitted openly in the packets/network and can thus be picked up by “snooping software”. Some vendors add extra functionality to limit only a subset of IP-addresses to access the SNMP Agent. For the TANDBERG Codec, there is not much damage that can be done, so the Codec will accept any request with a valid SNMP Community name.

6.2 Common, writeable MIB variables

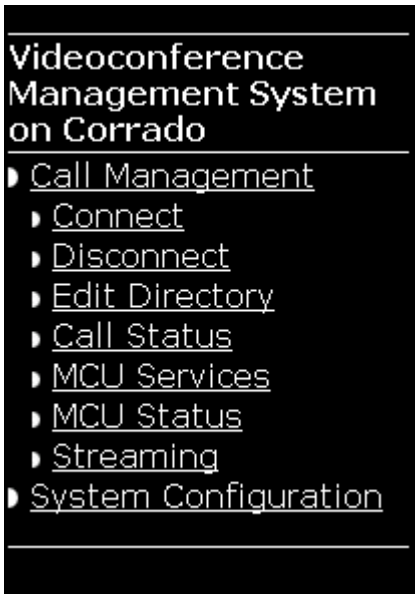
Two MIB variables will probably be the most likely to be set by users. Both of these are located in the system group of the mib-2 tree. The first, *System Location,* is meant to describe where this unit is located. The other, *System Contact*, can be used to register a named person who is responsible for this unit. *System Name* is the same as the MCU name for the Codec. This will also be preserved during a reboot, but should not be set via SNMP. By setting these variables the “most requested” information will be readily available and stored in the unit’s EEPROM (in other words, it will survive a restart or power off).

In addition, the SNMP Community Name, the trap receiver IP address and the status of the AuthenticationFailure trap will be stored, and survive a restart/power off of the Codec.

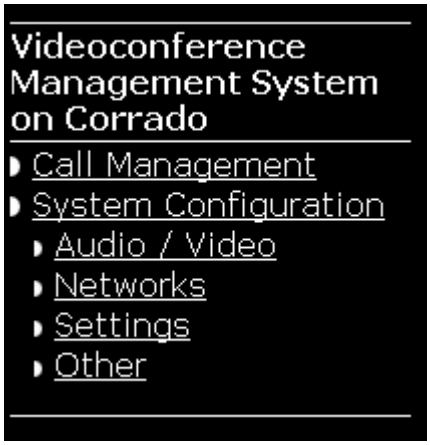
6.3 Configuring SNMP via the WWW interface

To configure SNMP by using the WWW interface, point your browser to your codec.

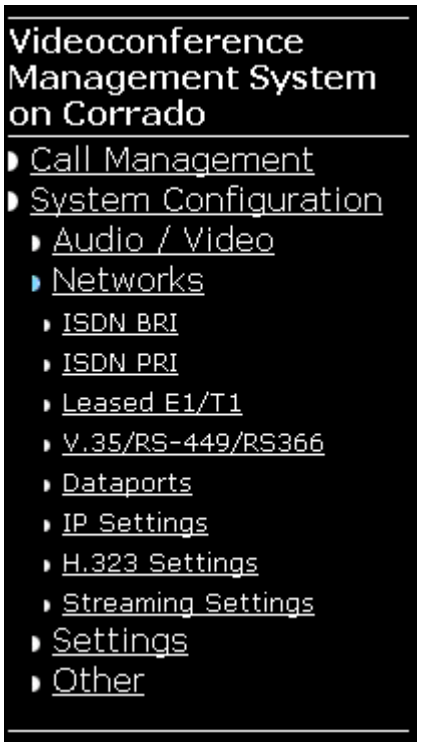
First select “*System configuration*”:



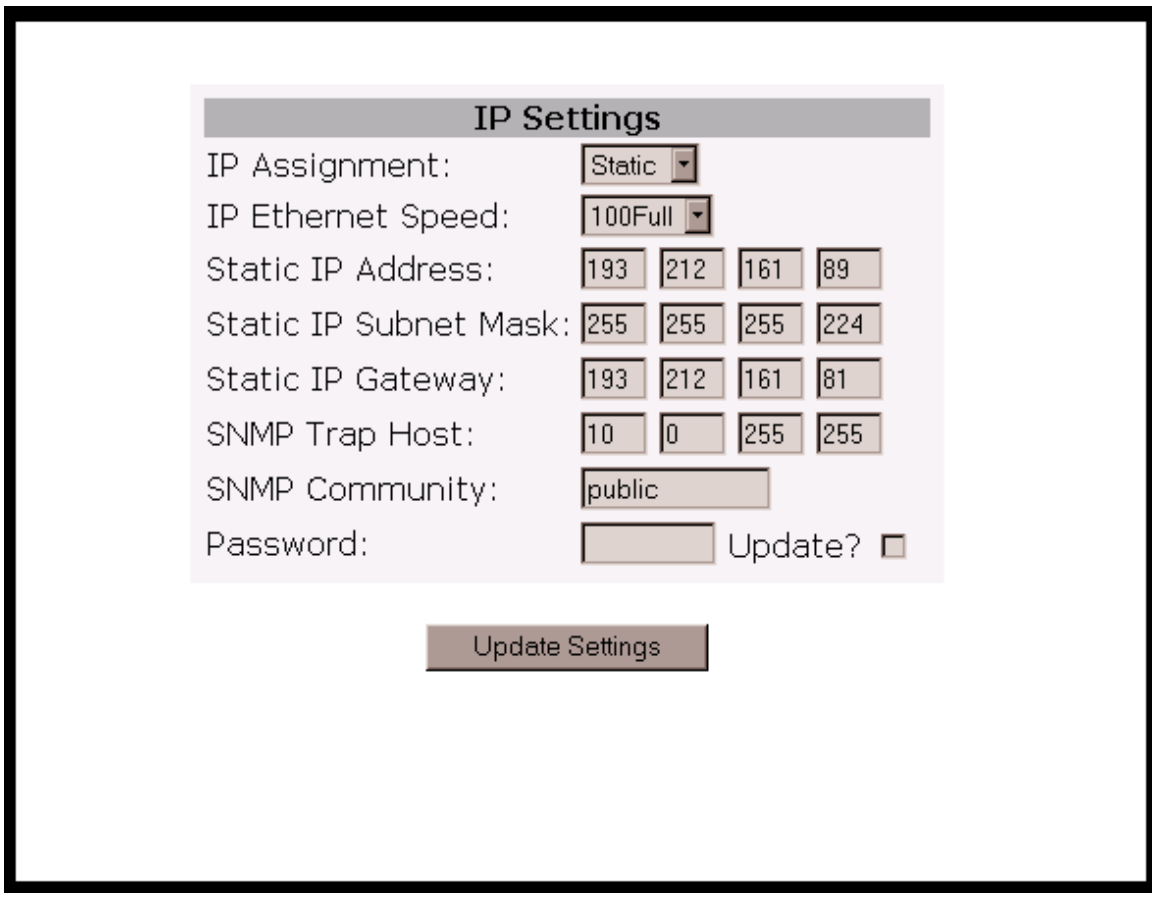
Then select *"Networks"*



Next, select "*IP Settings*"



You will then get the configuration window below:



The most important field to update is the *SNMP Community* string field. This is the SNMP "password" that are used to ensure that only allowed entities can access the SNMP information. This could be the same as for other networked units at your location, or another password of your selection. The default community name is "public", but this should be changed if SNMP is used.

If you have an SNMP Manager or some SNMP-trap enabled software, insert the IP-address of the host running this software in the "SNMP Trap Host" field. This will enable the Codec to send traps to this host. See the section about traps below.

6.4 Configuring SNMP via the dataport

Dataport commands are available by using "telnet" to connect to the codec. Use "telnet <ip-address>" (for example by selecting Start->Run in Windows).

All snmp configuration is available by the command "snmp".

This table will list all the options available for this command

Syntax	Example	Description
`snmp`	`snmp`	Lists the current settings for SNMP Community name, System Contact, System Location and SNMP Trap host (receiver).
`snmp cn <community name>`	`snmp cn topsecret`	Sets the SNMP Community Name to the given value
`snmp hi <trap host>`	`snmp hi 10.0.0.2`	Sets the IP-address of the host/program that can receive SNMP traps
`snmp sc <system contact>`	`snmp sc Robertson`	Sets the name of the contact person for this unit
`snmp sl <system location>`	`snmp sc MainOffice`	Sets the system location for this unit

The system contact and system location parameters can also be updated by using an SNMP Manager program. This is the same fields that are located in the mib-2 system group that is supported by the Codec.

6.5 Configuring SNMP via the remote control

Using recent (A2.0+) software releases you can configure the SNMP Community Name and SNMP Trap Host by the menu with your remote control. Go to "*Terminal Settings*" and "*LAN Settings*" to find these menu entries. Use your remote control to enter appropriate values.

7. TRAPS

7.1 What are traps

Traps are messages sent from the SNMP Agent (in the Codec) to some kind of managing software. Traps are indicators of an important event that has happened.

Traps that are sent from the Codec are of type coldStart or authenticationFailure (if enabled). WarmStart traps are not sent from the Codec, since the Agent will not be restarted in any other way than a hard reset/boot.

7.2 Enabling traps

By setting up the trap host in either the WWW interface or by using the dataport command, traps will be sent out. If one also requires authentication failure traps, these must be enabled by using some SNMP Manager (for example TSNMP has a dedicated button for this).

7.3 Authentication failure traps

Authentication failure traps are sent when the SNMP Agent receivers a valid SNMP request that contains a faulty SNMP Community Name. As mentioned earlier, the password protection of SNMP v1 is not very good, but by enabling these kinds of traps the manager can react if some “untrusted” party tries to access SNMP information in the Codec.

8. SNMP Testing software

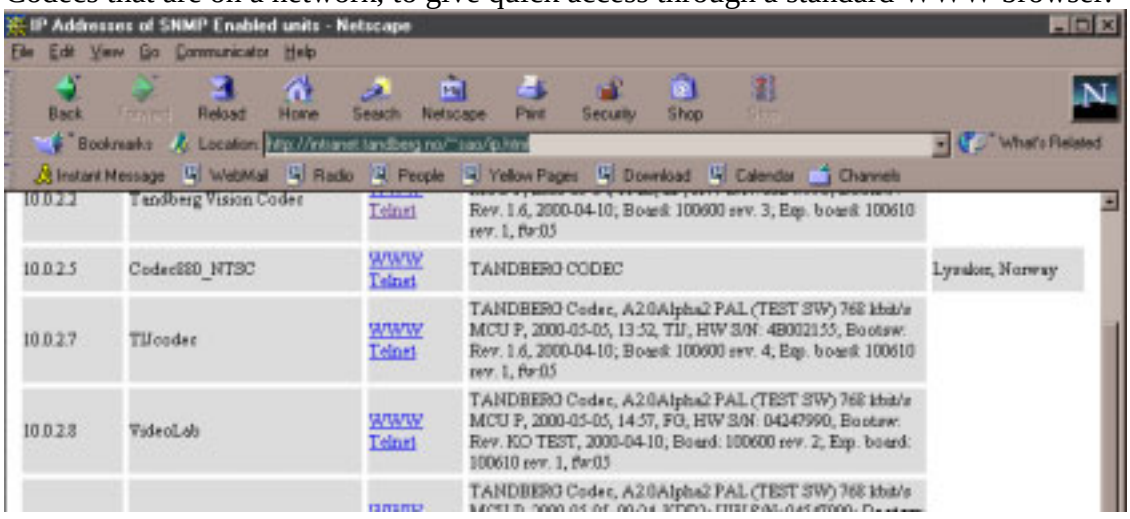
A few programs has been made to demonstrate the SNMP functionality in the Codec. TSNMP (TANDBERG SNMP Tool) is a simple mib-2 browser that can be used to enable or disable authentication traps, and scan through the supported MIB variables.

See the TSNMP Documentation for more details about this program.

A few other examples utilizing SNMP can also be obtained from TANDBERG.

9. Examples of SNMP usage

By using 3rd party SNMP tools, even open source ones, making small scale management applications can be done quite easily. This WWW application lists the Codecs that are on a network, to give quick access through a standard WWW browser.



By using Perl with an SNMP toolkit, and outputting standard HTML for any HTTP server, such applications can be made quickly.

Similar applications with Traps can be made to do surveillance if the Codecs are restarted.

10. SOFTWARE Version Differences

SOFTWARE VERSION	DESCRIPTION
Prior software releases	No SNMP support
C4.0 and A2.0	Contains RFC1157 SNMPv1 RFC1213 Mib2, no Enterprise MIB no Enterprise specific traps.
B1.0	Contains RFC1157 SNMPv1 Enterprise MIB with callMode and unitDesc Enterprise Trap # 1 callMode change
B2.0	Bug fixes only
B3.0	Much improved Enterprise MIB, with many new status variables. 3 new enterprise traps; genericError, downspeeding and low battery on remote control.
B4.0	Improved MIB and trap functionality. Extented the callTable and logTable. Added a few more entries in the codec part of the mib. Also added quite a few more traps.

11. TANDBERG Enterprise MIB

11.1 TANDBERG ENTERPRISE MIB ASN.1

```
-- Ignore this, sao@tandberg.no
-- TANDBERG-MIB { iso org(3) dod(6) internet(1) private(4)
enterprises(1) 5596 }
-- TANDBERG Telecom AS ENTERPRISE MIB, sao@tandberg.no
--

TANDBERG-MIB DEFINITIONS ::= BEGIN
IMPORTS
      enterprises, TimeTicks
      FROM RFC1155-SMI
      OBJECT-TYPE
      FROM RFC-1212
      DisplayString
      FROM RFC1213-MIB
      TRAP-TYPE
        FROM RFC-1215;


tandberg    OBJECT IDENTIFIER ::= { enterprises 5596 }
videoconf   OBJECT IDENTIFIER ::= { tandberg 10 }
status            OBJECT IDENTIFIER ::= { videoconf 3 }
callstatus  OBJECT IDENTIFIER ::= { status 2 }
logging           OBJECT IDENTIFIER ::= { videoconf 4 }
services        OBJECT IDENTIFIER ::= { videoconf 5 }
system            OBJECT IDENTIFIER ::= { videoconf 6 }
codec       OBJECT IDENTIFIER ::= { system 4 }
experimental      OBJECT IDENTIFIER ::= { videoconf 8 }

-- Common parameters

callMode OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Call Mode = Idle, Telephone, Videophone, DuoVideo, MultiSite,
... "
    ::= { callstatus 1 }

callLastError OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Descriptive error message, set after a Enterprise Error Trap
has been sent"
    ::= { callstatus 2 }

callLastErrorTime OBJECT-TYPE
    SYNTAX TimeTicks
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "System time (since boot) when last error message was set"
    ::= { callstatus 3 }

callLastCauseCode OBJECT-TYPE
    SYNTAX DisplayString
```

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "The cause value"
    ::= { callstatus 4 }

callLastCauseCodeLoc OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "The cause location"
    ::= { callstatus 5 }

statusStreaming OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Indicates if streaming is enabled or not. Idle or Streaming"
    ::= { services 2 }

statusVNC OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Indicates VNC status; idle, active, connectionfailure,
toomanyattempts, authfailure, resfailure"
    ::= { services 3 }

-- call information

callNumber OBJECT-TYPE
    SYNTAX INTEGER (0..1024)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Number of channels available for this unit"
    ::= { status 3 }

callTable OBJECT-TYPE
    SYNTAX SEQUENCE OF CallEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "The Channel information table"
    ::= { status 4 }

callEntry OBJECT-TYPE
    SYNTAX CallEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "Information about each channels status, activity and usage"
    INDEX { callID }
    ::= { callTable 1 }

CallEntry ::=
    SEQUENCE {
    callID
    INTEGER,
    callTag
    INTEGER,
    callRemoteSite
```

```
    DisplayString,
    callDirection
    DisplayString,
    callStatus
    DisplayString,
    callChannel
    DisplayString,
    callType
    DisplayString,
    callCause
    DisplayString,
    callDuration
    TimeTicks,
    callIncomingRestrict
    DisplayString,
    callOutgoingRestrict
    DisplayString,
    callAccNo
    DisplayString,
    callEncryption
    DisplayString
    }

callID OBJECT-TYPE
    SYNTAX INTEGER (0..1024)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Channel number"
    ::= { callEntry 1 }

callTag OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Unique call identifier since reboot"
    ::= { callEntry 2 }

callRemoteSite OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Telephone number or other address of remote connection"
    ::= { callEntry 3 }

callDirection OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle, incoming, outgoing"
    ::= { callEntry 4 }

callStatus OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle,syncing,capex,unframed,speech,disconn,synced"
    ::= { callEntry 5 }

callChannel OBJECT-TYPE
    SYNTAX DisplayString
```

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle,speech,h221-,bonding-,h323,unknown"
    ::= { callEntry 6 }

callType OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle,answering,calling,connected,disconnecting,disconnected"
    ::= { callEntry 7 }

callCause OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Cause code for failures"
    ::= { callEntry 8 }

callDuration OBJECT-TYPE
    SYNTAX TimeTicks
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Duration of this connection"
    ::= { callEntry 9 }

callIncomingRestrict OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "restrict,norestrict"
    ::= { callEntry 10 }

callOutgoingRestrict OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "restrict,norestrict"
    ::= { callEntry 11 }

callAccNo OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "The access code entered for the call "
    ::= { callEntry 12 }

callEncryption OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "encryption status"
    ::= { callEntry 13 }

-- start of inncoming entries

incomingNumber OBJECT-TYPE
```

```
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Number of entries in incomingTable"
    ::= { status 8 }

incomingTable OBJECT-TYPE
    SYNTAX SEQUENCE OF IncomingEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "Table with information about incoming connections"
    ::= { status 9 }

incomingEntry OBJECT-TYPE
    SYNTAX IncomingEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "Information about each channels status, activity and usage"
    INDEX { callIncomingId }
    ::= { incomingTable 1 }

IncomingEntry ::=
    SEQUENCE {
    callIncomingId
    INTEGER,
    callIncomingAudio
    DisplayString,
    callIncomingVideoMode
    DisplayString,
    callIncomingVideoResolution
    DisplayString,
    callIncomingDuoResolution
    DisplayString
    }


callIncomingId OBJECT-TYPE
    SYNTAX INTEGER ( 0..1024 )
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Identifier for each incoming table entry."
    ::= { incomingEntry 1 }

callIncomingAudio OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Incoming audio compression method"
    ::= { incomingEntry 2 }

callIncomingVideoMode OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Incoming video compression method"
    ::= { incomingEntry 3 }

callIncomingVideoResolution OBJECT-TYPE
    SYNTAX DisplayString
```

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Incoming resolution for video"
    ::= { incomingEntry 4 }

callIncomingDuoResolution OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Incoming resolution for duovideo"
    ::= { incomingEntry 5 }

-- end of incoming table

-- start of outgoing entries

outgoingNumber OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Number of entries in outgoingTable"
    ::= { status 10 }


outgoingTable OBJECT-TYPE
    SYNTAX SEQUENCE OF OutgoingEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "Table with information about outgoing connection"
    ::= { status 11 }


outgoingEntry OBJECT-TYPE
    SYNTAX OutgoingEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "entry in outgoing information table"
    INDEX { callOutgoingId }
    ::= { outgoingTable 1 }

OutgoingEntry ::=
    SEQUENCE {
    callOutgoingId
    INTEGER,
    callOutgoingAudio
    DisplayString,
    callOutgoingVideoMode
    DisplayString,
    callOutgoingVideoResolution
    DisplayString,
    callOutgoingDuoResolution
    DisplayString
    }

callOutgoingId OBJECT-TYPE
    SYNTAX INTEGER ( 0..1024 )
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Identifier for each outgoing table entry"
```

```
    ::= { outgoingEntry 1 }

callOutgoingAudio OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Outgoing audio compression method"
    ::= { outgoingEntry 2 }


callOutgoingVideoMode OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Outgoing video compression method"
    ::= { outgoingEntry 3 }

callOutgoingVideoResolution OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Outgoing resolution for video; qcif,cif,icif,..."
    ::= { outgoingEntry 4 }

callOutgoingDuoResolution OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Outgoing resolution for duovideo when enabled,
qcif,cif,icif,..."
    ::= { outgoingEntry 5 }

-- end of outgoing entries

-- Codec descriptors definition

    unitDesc OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "Type of Codec, version"
      ::= { codec 1 }

    versionId OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "Codec software configuration identifier for major
releases"
      ::= { codec 2 }

    softwareType OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "Software type descriptor (same as data port AT-command
ati3)"
      ::= { codec 3 }
```

```
    hardwareConfig OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "Codec hardware configuration (same as data port AT-command
ati6) "
      ::= { codec 5 }

    softwareConfig OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "Codec software configuration (same as data port AT-command
ati7) "
      ::= { codec 6 }

    localISDN OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "ISDN telephone number for this codec"
      ::= { codec 8 }

    dispbox OBJECT-TYPE
      SYNTAX INTEGER (0..1024)
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "User response from displaybox"
      ::= { codec 9 }

    protect OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "protect on/off on the dataport"
      ::= { codec 10 }

    ippass OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "password on/off on the dataport"
      ::= { codec 11 }

    h323alias OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
          "displays the h323alias of the codec"
      ::= { codec 12 }

-- end of generic definitions

-- logging of calls

logNumber OBJECT-TYPE
    SYNTAX INTEGER (0..1024)
```

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "The number of log entries in the logTable "
    ::= { logging 4 }

logTable OBJECT-TYPE
    SYNTAX SEQUENCE OF LogEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "The call log information table"
    ::= { logging 5 }

logEntry OBJECT-TYPE
    SYNTAX LogEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
      "A call log entry in the logTable "
    INDEX { logID }
    ::= { logTable 1 }

LogEntry ::=
    SEQUENCE {
    logID
    INTEGER,
    logTag
    INTEGER,
    logRemoteSite
    DisplayString,
    logDirection
    DisplayString,
    logStatus
    DisplayString,
    logChannel
    DisplayString,
    logType
    DisplayString,
    logCause
    DisplayString,
    logDuration
    TimeTicks,
    logIncomingRestrict
    DisplayString,
    logOutgoingRestrict
    DisplayString,
    logAccNo
    DisplayString,
    logEncryption
    DisplayString
    }

logID OBJECT-TYPE
    SYNTAX INTEGER (0..1024)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Identifier for each log table entry"
    ::= { logEntry 1 }

logTag OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
```

```
    DESCRIPTION
      "Unique log identifier since reboot"
    ::= { logEntry 2 }

logRemoteSite OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "telephone number, H323-Alias or IP-address of remote
connection"
    ::= { logEntry 3 }

logDirection OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle, incoming, outgoing"
    ::= { logEntry 4 }

logStatus OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle,syncing,capex,unframed,speech,disconn,synced"
    ::= { logEntry 5 }

logChannel OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle,speech,h221-,bonding-,h323,unknown"
    ::= { logEntry 6 }

logType OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "idle/answering/logging/connected/disconnecting/disconnected"
    ::= { logEntry 7 }

logCause OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Cause code for failures"
    ::= { logEntry 8 }

logDuration OBJECT-TYPE
    SYNTAX TimeTicks
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "Duration of this connection"
    ::= { logEntry 9 }

logIncomingRestrict OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
```

```
    DESCRIPTION
      "restrict, norestrict"
    ::= { logEntry 10 }

logOutgoingRestrict OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "restrict, norestrict"
    ::= { logEntry 11 }

logAccNo OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "The access code/account code used for the connection. 0 if
nothing was
used"
    ::= { logEntry 12 }

logEncryption OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
      "encryption status"
    ::= { logEntry 13 }

-- These TRAPS are commented out due to problems with some
-- MIB compilers. They are all valid.


--
-- callModeChange TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "Informs of change in Call Mode, whenever a codec goes
from Idle to Telephone, Videophone, MCU or similar."
--    ::= 1
--

-- genericError TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "callLastError has been set with a new message,
indicating some kind of Codec problem"
--    ::= 2
--

-- lowBattery TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "Battery is low in the remote control"
--    ::= 3

-- downSpeeding TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "Downspeeding, adjusting connection speed due to problem"
--    ::= 4

-- startupgrade TRAP-TYPE
--    ENTERPRISE
```

```
--    DESCRIPTION
--          "A codec software upgrade has begun"
--    ::= 5

-- finsishedupgrade TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "A codec software upgrade has been completed"
--    ::= 6

-- dispbox TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "When a display box (dispbox) is shown on the codec, and
the user
-- presses a quick-key, this trap is sent to indivate the quick-key
selected.
-- Values are 1,2,3"
--    ::= 7

-- connect TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "The call is connected"
--    ::= 8

-- encryption TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "Encryption status changed to des"
--    ::= 9

-- gatekeeper_reg_failure TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "gatekeeper registration failure"
--    ::= 10

-- gatekeeper reg_success TRAP-TYPE
--    ENTERPRISE
--    DESCRIPTION
--          "gatekeeper registration successs"
--    ::= 11

END
```

11.2 TANDBERG ENTERPRISE MIB TREE STRUCTURE

```
# TANDBERG-MIB registration tree (generated by smidump 0.2.14)

--iso(1)
  |
  +--org(3)
     |
     +--dod(6)
        |
        +--internet(1)
           |
           +--private(4)
              |
              +--enterprises(1)
                 |
                 +--tandberg(5596)
                    |
                    +--videoconf(10)
                       |
                       +--status(3)
                       |  |
                       |  +--callstatus(2)
                       |  |  |
                       |  |  +-- r-n DisplayString callMode(1)
                       |  |  +-- r-n DisplayString callLastError(2)
                       |  |  +-- r-n TimeTicks
callLastErrorTime(3)
                       |  |  +-- r-n DisplayString
callLastCauseCode(4)
                       |  |  +-- r-n DisplayString
callLastCauseCodeLoc(5)
                       |  |
                       |  +-- r-n Integer32 callNumber(3)
                       |  |
                       |  +--callTable(4)
                       |  |  |
                       |  |  +--callEntry(1) [callID]
                       |  |     |
                       |  |     +-- r-n Integer32     callID(1)
                       |  |     +-- r-n DisplayString callTag(2)
                       |  |     +-- r-n DisplayString
callRemoteSite(3)
                       |  |     +-- r-n DisplayString
callDirection(4)
                       |  |     +-- r-n DisplayString callStatus(5)
                       |  |     +-- r-n DisplayString callChannel(6)
                       |  |     +-- r-n DisplayString callType(7)
                       |  |     +-- r-n DisplayString callCause(8)
                       |  |     +-- r-n TimeTicks     callDuration(9)
                       |  |     +-- r-n DisplayString
callIncomingRestrict(10)
                       |  |     +-- r-n DisplayString
callOutgoingRestrict(11)
                       |  |     +-- r-n DisplayString callAccNo(12)
                       |  |     +-- r-n DisplayString
callEncryption(13)
                       |  |
                       |  +-- r-n Integer32 incomingNumber(8)
                       |  |
                       |  +--incomingTable(9)
                       |  |  |
```

```
                        |  |  +--incomingEntry(1) [callIncomingId]
                        |  |     |
                        |  |     +-- r-n Integer32
callIncomingId(1)
                        |  |     +-- r-n DisplayString
callIncomingAudio(2)
                        |  |     +-- r-n DisplayString
callIncomingVideoMode(3)
                        |  |     +-- r-n DisplayString
callIncomingVideoResolution(4)
                        |  |     +-- r-n DisplayString
callIncomingDuoResolution(5)
                        |  |
                        |  +-- r-n Integer32 outgoingNumber(10)
                        |  |
                        |  +--outgoingTable(11)
                        |     |
                        |     +--outgoingEntry(1) [callOutgoingId]
                        |        |
                        |        +-- r-n Integer32
callOutgoingId(1)
                        |        +-- r-n DisplayString
callOutgoingAudio(2)
                        |        +-- r-n DisplayString
callOutgoingVideoMode(3)
                        |        +-- r-n DisplayString
callOutgoingVideoResolution(4)
                        |        +-- r-n DisplayString
callOutgoingDuoResolution(5)
                        |
                        +--logging(4)
                        |  |
                        |  +-- r-n Integer32 logNumber(4)
                        |  |
                        |  +--logTable(5)
                        |     |
                        |     +--logEntry(1) [logID]
                        |        |
                        |        +-- r-n Integer32     logID(1)
                        |        +-- r-n DisplayString logTag(2)
                        |        +-- r-n DisplayString
logRemoteSite(3)
                        |        +-- r-n DisplayString logDirection(4)
                        |        +-- r-n DisplayString logStatus(5)
                        |        +-- r-n DisplayString logChannel(6)
                        |        +-- r-n DisplayString logType(7)
                        |        +-- r-n DisplayString logCause(8)
                        |        +-- r-n TimeTicks     logDuration(9)
                        |        +-- r-n DisplayString
logIncomingRestrict(10)
                        |        +-- r-n DisplayString
logOutgoingRestrict(11)
                        |        +-- r-n DisplayString logAccNo(12)
                        |        +-- r-n DisplayString
logEncryption(13)
                        |
                        +--services(5)
                        |  |
                        |  +-- r-n DisplayString statusStreaming(2)
                        |  +-- r-n DisplayString statusVNC(3)
                        |
                        +--system(6)
                        |  |
                        |  +--codec(4)
                        |     |
```

```
|     +-- r-n DisplayString unitDesc(1)
|     +-- r-n Integer32     versionId(2)
|     +-- r-n DisplayString softwareType(3)
|     +-- r-n DisplayString hardwareConfig(5)
|     +-- r-n DisplayString softwareConfig(6)
|     +-- r-n DisplayString localISDN(8)
|     +-- r-n Integer32     dispbox(9)
|     +-- r-n DisplayString protect(10)
|     +-- r-n DisplayString ippass(11)
|     +-- r-n DisplayString h323alias(12)
|
+--experimental(8)
```