IEC 62379-1

Edition 1.0 2007-08

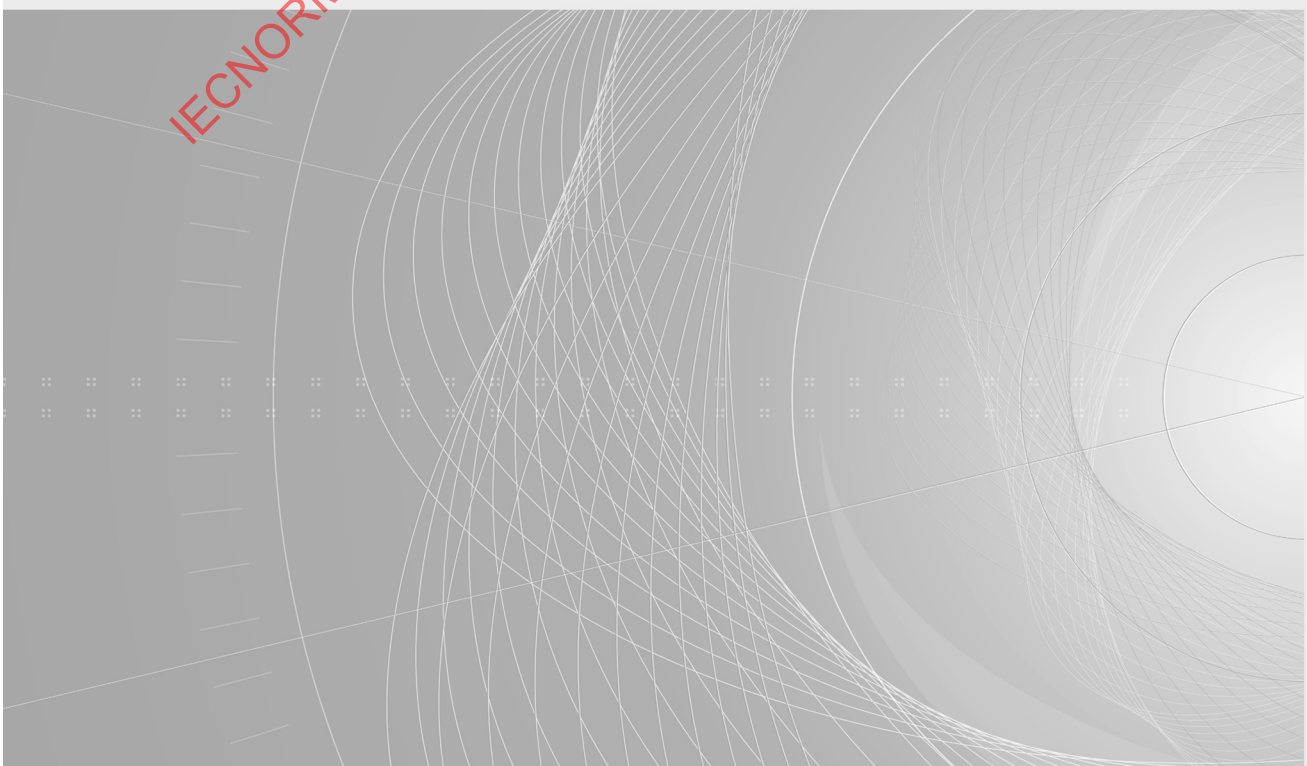
INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Common control interface for networked digital audio and video products – Part 1: General

Interface de commande commune pour produits audio et vidéo numériques connectés en réseaux –
Partie 1: Généralités



IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 62379-1

Edition 1.0 2007-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Common control interface for networked digital audio and video products – Part 1: General

Interface de commande commune pour produits audio et vidéo numériques connectés en réseaux – Partie 1: Généralités

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XB
CODE PRIX

ICS 33.160; 35.100

ISBN 978-2-83220-239-5

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMON CONTROL INTERFACE FOR NETWORKED DIGITAL AUDIO AND VIDEO PRODUCTS –

Part 1: General

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62379-1 has been prepared by IEC technical committee 100: Audio, video and multimedia systems and equipment.

This bilingual version (2012-08) corresponds to the monolingual English version, published in 2007-08.

The text of this standard is based on the following documents:

FDIS	Report on voting
100/1248/FDIS	100/1281/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The French version of this standard has not been voted upon.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

0 Introduction

0.1 Overview

This family of standards specifies a control framework for networked audiovisual equipment.

It provides a means for management entities to control not only transmission across the network but also other functions within interface equipment.

Although it was originally developed for audio over asynchronous transfer mode (ATM) in radio broadcasting, the control framework has been extended to encompass video and other time-critical media, as well as other networking technologies and other applications in both professional and consumer environments.

The control framework provides a number of key features:

- it provides a consistent interface to the functionality in an audiovisual unit;
- it enables systems to be built that are truly "plug and play", by providing the means for equipment to discover what units are connected to the network and what their capabilities are;
- it links discrete areas or blocks of functionality together in a consistent and structured way;
- it allows us to define small focused building blocks from which more complex functionality can be built;
- it ensures new functionality can be developed and integrated consistently and easily into the framework.

The functionality provided by an audiovisual unit is represented using one or more "blocks" (such as a cross-point switch or a gain control), structured and connected together using the control framework.

As a further aid to the "plug-and-play" functionality, a common format for audio and video being conveyed across the network is also specified, to avoid situations in which two pieces of equipment fail to communicate because there is no format which both support. Equipment may, of course, also support other formats appropriate to particular applications, and the standard mechanisms for initiating and terminating communication will work for those formats in the same way as for the standard formats.

0.2 Structure of the family of standards

IEC 62379 is intended to include the following parts:

Part 1: General

Part 2: Audio

Part 3: Video

Part 4: Data

Part 5: Transmission over networks

Part 6: Packet transfer service

Part 1 specifies aspects which are common to all equipment.

Parts 2 to 4 specify control of internal functions specific to equipment carrying particular types of media; in the case of Part 4, this would be time-critical media other than audio and video, for instance, RS232 and RS422 data for applications such as machine control, or the state of

the “on air” light in a broadcast studio. Part 4 does not refer to packet data such as the control messages themselves.

Part 5 specifies control of transmission of these media over each individual network technology, with a separate subpart for each technology. It includes network specific management interfaces along with network specific control elements that integrate into the control framework.

Part 6 specifies carriage of control and status messages and non-audiovisual data over transports that do not support audio and video, such as RS232 serial links, with (as with Part 5) a separate subpart for each technology.

0.3 Model of the equipment being controlled

0.3.1 Blocks

A piece of equipment (a "unit") is regarded as being composed of functional elements or "blocks" which may be linked to each other through internal routing.

Blocks may have inputs, outputs and internal functionality. In general, the output of one block connects to the input of the next block in the processing chain. Blocks can have some associated control parameters and/or status monitoring accessible via the control framework management interface.

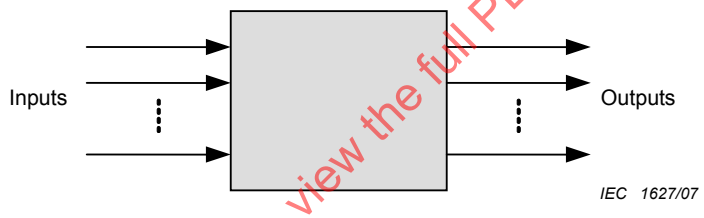


Figure 1 – A block

A typical block would be a pre-amplifier, which has one input, one output, and a parameter which sets the gain. Another would be a mixer, with several inputs, one output, and parameters to select the contribution of each input to the mix; these parameters are effectively fader settings. A tone generator would have one output and no inputs, and parameters that set the level, frequency, etc.

There is a special class of blocks called "ports"; ports provide an external connection to other equipment. An "input port" is one where audio, video, or other data enters the unit and an "output port" is one where it leaves the unit. Sometimes the port corresponds to a physical connection, for instance, an XLR socket for analogue audio; sometimes it is a virtual entity which can be one end of a connection across a network, or one channel on an interface such as AES10 (MADI) which conveys multiple audio streams.

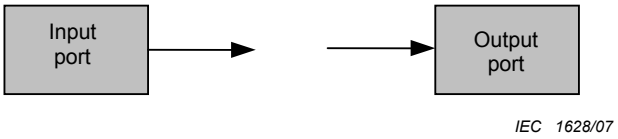


Figure 2 – Ports

An input port has no inputs (or rather, no internal inputs; it will have an external input, but that is not part of the model of the internal structure of the unit) and a single output, which

supplies the incoming stream to the inputs of other blocks. In the case of a network port, parameters would specify the network address; a physical audio port might have parameters which show the sampling rate and bit depth. Similarly, an output port has a single input and no (internal) outputs.

Figure 3 shows an example of how the various blocks connect together within a unit. Note that each input is connected to exactly one output, but an output may be connected to several inputs, or to none.

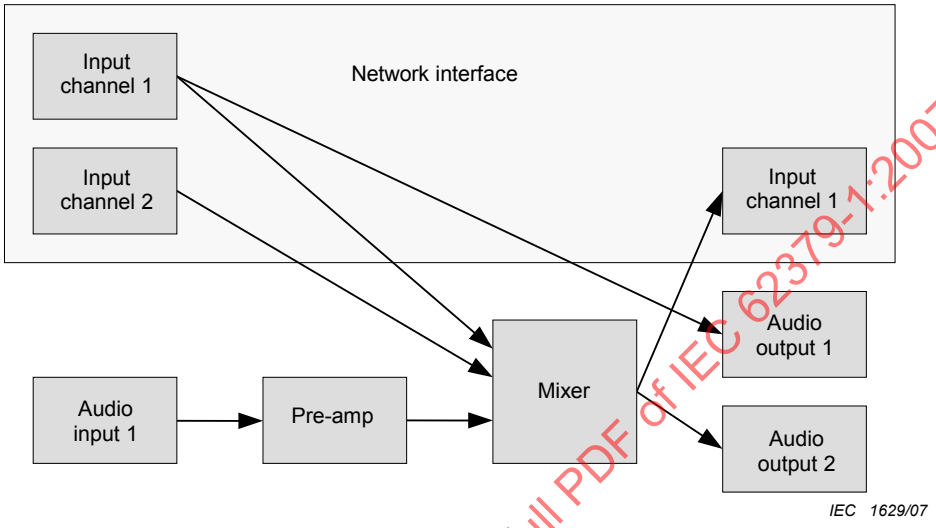


Figure 3 – Example of a "unit"

There is a block which performs a mix between three inputs, two from the network and one from a physical audio port (or, looking at it another way, two from remote sources and one from a local source). The local source is connected via a pre-amplifier. The resulting signal is output locally at output 2 and also transmitted on the network. There is another local output which carries a copy of one of the remote sources.

The set of available blocks, the connectivity between them, and the parameter settings for each may be fixed, or changeable by a management terminal, or read-only but changeable by external factors. Where blocks are implemented in software, a unit may provide the ability for a management terminal to create and delete them. Where blocks are implemented in physical hardware, the blocks themselves cannot be changed but it may still be possible for the management terminal to reprogramme the routing between them.

0.3.2 Control framework

The control framework consists of two lists; a list of blocks (also called control elements), and a list of connections between them. In both lists, an individual block is identified by a "block id", which is a number that is different for each block in a unit.

A block's entry in the list of blocks shows what type of block it is, represented by a globally unique value as described in 0.3.5.

Groups of blocks that are connected together are called processing chains. A processing chain typically represents what a unit does as a whole, so, for instance, a unit that alters the gain of an input to produce an output would have one simple processing chain that consists of an input port connected to a gain block which is connected to an output port.

0.3.3 How the framework helps when designing units

The standard anticipates that many control blocks will be designed and implemented over time to control the many different sorts of functionality audio and audiovisual units provide.

Units can be built from existing blocks or new ones created as required. It will often be possible to represent complex, product-specific control functionality using a number of linked instances of simpler, standard blocks that together provide the functionality required.

0.3.4 How the framework enables "plug and play"

A management terminal simply needs to recognize those blocks that are relevant to the functions it controls. (The term "management terminal" covers a wide variety of equipment, from a broadcast control system to the user interface of a device on a home network.)

It can discover what units are present on the network and what functions each contains; it does not need to recognize the units themselves, only the blocks that describe the functionality in which it is interested.

The discovery process would be:

- to create a list of the units, beginning with those to which it is directly connected; units can be uniquely identified by their 48-bit MAC address;
- to retrieve the list of blocks from each device on the list; if any are network ports which give access to further devices, to add those devices to the list (unless they are already on it);
- to retrieve the connectivity between any blocks for which it is relevant.

For instance, the user interface to a surround-sound audio system might search for units containing audio sources, find those for which a processing chain exists that allows them to be made available to the user, and offer them in a menu. It would also identify functions in the processing chain such as volume control and play-out controls (pause, rewind, skip track, etc.).

In a radio broadcast control system, a similar process could be performed when the system is installed and at any time when equipment is added or replaced. This process would be under the control of the installer, rather than occurring automatically, but should at least relieve the installer of the necessity to type in network addresses.

0.3.5 Defining a new type of block

A block's entry in the block list shows what type of block it is, represented by an object identifier (OID) (see 0.4.2) which is a globally unique value that identifies the block type definition.

The main requirement when adding a new type of block to the control framework is for its block type definition to follow the conditions below:

- the globally unique OID identifies a MIB table or group of MIB tables, with each table containing a variable number of rows.
- the table(s) are indexed using the block id to access control objects associated with individual instances of this block type.

In effect, the framework provides the entry point to controlling each block of functionality. The actual details of how to control that functionality will always need to be specified individually.

0.4 Management information base (MIB)

0.4.1 Objects

Communication between the management terminal and the managed unit is by the simple network management protocol (SNMP), which defines all management operations in terms of reads and writes on a hierarchically organized collection of variables, or "managed objects", known as a MIB.

Each object is identified by an OID which is a sequence of numbers; in the descriptions they are separated by dots, and there are also alphanumeric names which can be used instead. Identifiers of objects defined in one of this family of standards begin 1.0.62379.*p* if defined in part *p*, or 1.0.62379.*p.s* if defined in part *p-s*.

Each block is described by a group of objects (a "record"). These objects are the control parameters and status variables. For each type of block, the structure of the record is specified in one of the parts of IEC 62379 or (for product-specific or application-specific blocks) elsewhere. The connectivity is described by a table containing an identification of the output to which each input is connected (see `connectorTable` in 4.2.5).

Thus, the management terminal can discover what functionality a unit has and may be able to reprogramme some of it. The SNMP protocol dictates that a command to change an object is always confirmed by a message showing the new value of the object, so if a unit does not support the full range of possible values of a parameter it can choose the one nearest to the requested value and will report back to the management terminal exactly what it has done.

0.4.2 Other uses of OIDs

Sometimes OIDs are used to identify things other than objects. Each block type is represented by an OID; in this case, it is also the root (the first few numbers in the sequence) of the OIDs for all the objects in the record that describes each block of that type.

The OIDs are not confined to those specified in the various parts of IEC 62379; for product-specific blocks, implementers may define other types with OIDs rooted at, for example, 1.3.6.1.4.1.*n*, where *n* is the enterprise number of the manufacturer of the unit. A general-purpose management terminal which only recognized the standard OIDs would see these product-specific blocks as "black boxes"; it would recognize their connectivity but not be able to control or monitor their operation.

Media formats (audio, video, etc.) are also identified by OIDs. Here, again, OIDs not rooted at 1.0.62379 may be used for formats that are not defined in this family of standards; provided the sending and receiving devices both support the format, a call can be set up without the management terminal being able to recognize it, though the management terminal might not be able to display a user-friendly description of it.

0.4.3 Migration to XML

Increasingly, XML is being used as the interface to network management applications. However, communication with the management agents in the networked devices is usually through a gateway that translates between SNMP and XML, so that the managed unit only needs to support SNMP.

A future addition to IEC 62379 (amendment or additional part) might standardize the XML form; however, in that event, the underlying managed objects would remain the same in both forms.

0.5 Status broadcasts

0.5.1 Introduction

Status pages are messages containing structured values representing some internal state of a unit. Each page is organized into a fixed format of related information. A unit may define and support multiple types of status page.

Related status pages are collected together into groups called status broadcast groups. When a remote entity requests a status broadcast, it specifies which group it wants to receive. Status broadcast groups are identified by OIDs.

Status pages are the preferred method for regularly monitoring the state of a unit. Polling fields in the MIB put more load on both the unit and the network, because they require the unit to process an SNMP request on every occasion, rather than just once to set up the broadcast. If the same information is required at multiple locations, there will be a separate request from each, and multiple copies of the information must be sent, whereas with the broadcast only one copy is sent, being duplicated by the network as required to reach all its destinations.

This standard defines a number of status pages and groups. Other parts define their own status pages and groups. Manufacturers may also define product-specific status pages and groups.

0.5.2 Status page information sources

Status pages and groups may be associated either with a single block or with the unit as a whole.

Three pieces of information are required to initiate a status page broadcast call (other information may be required; however, this will be network-specific and specified in the relevant subpart of Part 5):

- the block id – of the block that is to be the source of the status broadcast group call (a null value is used for status broadcasts associated with the unit as a whole);
- the page group OID – of the particular status broadcast group to be produced;
- the required page rate – the rate at which the pages are generated.

If a unit supports multiple instances of a block type (for instance, an AES3 audio output, which supports an audio level status broadcast group), it is not required to implement the associated status broadcast group for each instance – it may implement it for just a subset of them.

0.5.3 Status page general format

The first two octets of a status page indicate the page type. Page type identifiers are required to be unique for all pages that are part of a given page group, but otherwise may be freely allocated by the organization or manufacturer that specifies the page content. The format of the rest of the page depends on the page type; the maximum length is 484 octets.

Numbers are coded in binary using a fixed number of bytes and big-endian. Text strings are in UTF-8.

0.6 Calls

The network is expected to provide the following two kinds of services.

- Fixed bit rate one-to-many service for media streams and status broadcasts, with guaranteed latency and throughput.
- "Best-effort" bidirectional one-to-one packet transfer service for control messages.

On an ATM network, these map onto CBR point-to-multipoint and UBR point-to-point calls, respectively.

On a connectionless (and stateless) network such as IP, there is no direct equivalent of the concept of an ATM "call". However, if the network is to provide the necessary quality of service (QoS) guarantees for media streams, there will need to be some kind of mechanism for allocating the resources needed for a stream. This mechanism must, in many ways, be similar to the call set-up process on connection-oriented networks.

In the case of the packet transfer service, which may well be connectionless at the network layer, for many applications there will be a relationship between the management terminal and the managed unit within which some "state" information persists between transactions. For instance, when updating software in the unit (see 0.9), only one management terminal can write a given memory area at a time. Thus a "session" will exist between the management terminal and the managed unit, which corresponds to a call on a connection-oriented network.

Associated with any media stream is "call identity" information which can include information about the stream itself, the point where it enters the system, and each destination. Destinations have an "importance" assigned to them, and it is normally only the most important destinations that are reported. More details are given in Clause A.4.

0.7 Privilege levels

Throughout IEC 62379, the concept of "privilege levels" is used to distinguish different kinds of user. This concept was developed for radio broadcasting studios, and the names of the levels reflect that but will also be useful in other applications.

Four privilege levels are defined:

- listener;
- operator;
- supervisor;
- maintenance.

Listener is the lowest privilege level, intended for use by those who wish to monitor audio or video signals passing through the unit (for example, someone who wishes to listen in on the output of a studio via their PC). Listeners can set up calls from audio and video sources to equipment which is local to them but cannot change anything that would affect the experience of other users.

Operator is the next privilege level, intended for use by those who are controlling the day-to-day operation of the unit (for example, a studio technical operator). Operators can change things which affect other users, such as the mix of signals that provides the output from a studio, or by issuing "pause", "seek", etc. commands to play-out equipment.

Supervisor is the next privilege level, intended for use by those who are controlling and maintaining the network such as a control room technical operator.

Maintenance is the highest privilege level, intended for use by those who need to perform tasks that might disrupt normal operation of the unit, such as updating firmware or causing the unit to enter a diagnostic mode.

Privilege levels are intended to be used for two purposes. The first is to allow management call capacity to be reserved for each category of caller, to prevent important management calls being blocked because too many lower-level calls are in progress. The second is to prevent callers from performing inappropriate control tasks, by limiting the commands accepted at lower levels.

To enable the latter functionality, every call has a privilege level associated with it, and a call may not be set up, modified, or torn down by a management call of lower privilege. For instance, an operator can set up calls with operator privilege which then cannot be affected by management calls that only have listener privilege, although they can be modified or torn down by other operators and by supervisors. Supervisors can set up calls which neither listeners nor operators can disturb; the connection between a continuity studio and the transmission chain might be an example.

This specification requires that at least one management call must be accepted at each privilege level at any given time. In practice, the call capacity that needs to be reserved for each level is likely to depend on the context within which the unit is used, so a means of configuring this is also specified. It is intended that any unreserved call capacity will be allocated on a first-come first-served basis.

0.8 Automation

The automation mechanism allows for single or multiple values to be set at a given time. The actual scheme uses a list of automation events where each event consists of a time, the OID of an object in the MIB, and the value to put in it at that time.

This removes the uncertainty introduced by the best-effort service on the network; the controller can add the event to the table far enough in advance to give time to repeat the request if it is lost. Also, it can programme a number of events to occur simultaneously.

Also, multiple events can be associated so they occur one after the other much like a macro.

0.9 Uploading software

This standard includes (in 4.2.8 and 5.2) a mechanism for updating software and other configuration information in units supporting the common control interface.

The intention is for a management terminal to be able to update software in a large number of different pieces of equipment from different manufacturers without needing to run manufacturer-specific applications.

The MIB objects defined in 4.2.8 are intended to provide a model which is common to all the various kinds of memory that might be used in the managed unit. A unit may contain more than one “class” of memory; different classes may be physically different, for instance, flash memory and rotating magnetic memory, and/or reserved for different kinds of data, for instance, software and audio clips.

EXAMPLE 1: Simple system using flash memory

Flash memory is composed of blocks, typically 64K bytes each. An individual byte can be written provided this does not involves changing any bit from 0 to 1. A whole block can be “erased”, after which every bit in the block is a 1.

Each area consists of either a single block or several adjacent blocks; a few bytes are reserved to hold the length, type, and serial number. The "handle" which identifies an area is the high part of the address of the first byte of the area.

Deletion is by erasing all the blocks that comprise the area, which may take several seconds for some flash memory chips. After deletion, if there is an adjacent empty area the two areas are merged to form a single empty area (so one of them will disappear from the table).

If there is no other memory into which components can be loaded, all areas should be class 1.

EXAMPLE 2: Disc with filing system

Each file is an “area”, and there is another (single) area containing all the free space. In the case of a Unix filing system, the "handle" might be the file's inode number.

If the product has both disc and flash, it might identify the flash as class 2 and the disc as class 1; or it might divide the disc into separate partitions identified as classes 3, 4, etc.

One object for each area shows the `access` permitted, i.e. whether it may be written and/or erased. Whether an area is included in the table, and the access permitted if so, may depend on the privilege level. The management terminal may be able to change the `access`, but usually this will be controlled by the managed unit; for instance, it may set all areas required to load the software currently running to read-only (i.e. not writable), and the rest to full access, so that new software can be uploaded into currently unused areas, but the current software cannot be overwritten until the new software has been completely loaded.

Another is the `status` which shows what operation is being performed on the area. The management terminal requests deletion of an area by setting its `status` to “erasing”; the managed unit will then delete its current contents and set the `status` to “empty”. It may then amalgamate it with another empty area in the same class of memory.

An area also has a `serialNumber`; new software is given the serial number next in sequence after that for the current software. When the unit is restarted, the (valid) software with the most recent serial number is loaded; the procedures in 5.2.3 ensure that partly loaded software will not be valid, for instance, if the unit is restarted before the uploading process is complete. Thus, if the unit is restarted before the new software has been completely loaded, the old software will run; if after, the new software will run.

Two methods of software distribution are supported. The software may be supplied as a package, for instance, on a CD or by e-mailing a ZIP file, as specified in 5.2.6.1; when new software is available, a new package must be distributed. Alternatively, a "product file" containing a URL may be supplied, and the software downloaded over the Internet as specified in 5.2.6.2. It is made easy for the management terminal to check whether new software is available.

0.10 Encapsulation of messages

Throughout this family of standards, the word "message" is used to mean an octet string which is conveyed across the network as a single unit. On an IP network, it will be the "data" part of a packet or datagram, i.e. excluding all the headers and framing. In the case of an ATM network, it will normally be an AAL5-SDU.

Thus on an IP network SNMP messages are packaged in the normal way for SNMP, i.e. using UDP over IP. On an ATM network they are packaged directly in AAL5, in the same way as in ILMI.

0.11 Further information

Additional explanations of some aspects of this standard are given in Annex A.

COMMON CONTROL INTERFACE FOR NETWORKED DIGITAL AUDIO AND VIDEO PRODUCTS –

Part 1: General

1 Scope

This part of IEC 62379 specifies a control interface for products which convey audio and/or video across digital networks.

Separate documents specify items specific to a particular type of traffic, a particular networking technology, or a particular class of application.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*

ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

IEEE Std 802:2001, *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*

RFC1157, *Simple Network Management Protocol (SNMP)* (IETF Standard #15 STANDARD)

RFC 1441, *Introduction to version 2 of the Internet-standard Network Management Framework (IETF)*

RFC 3411-3418, *Simple Network Management Protocol version 3* (IETF Standard #62)

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
coordinated universal time
UTC
time scale which forms the basis of a coordinated dissemination of standard frequencies and time signals (see ITU-R Recommendation TF.460)

3.2
call
either a point-to-point call or a point-to-multipoint call

3.3
call replacement
process whereby a new call is set up to replace an existing call, the new call taking a different route through the network, and the destination changes from receiving cells on the old call to receiving them on the new call without interruption to the flow of data carried by them

3.4
global positioning system
GPS
globally available timing data source from which UTC can be derived; also a navigational aid

3.5
MAC address
48-bit LAN MAC address as specified in 9.2 of IEEE 802, identifying a point of attachment of a unit to a network

3.6
message
octet string which is conveyed across the network as a single unit

3.7
non-volatile real-time clock
component which provides the time of day and date and continues to do so in the absence of an external reference, even if the unit which contains it has not been continuously powered since the last time an external reference was available

3.8
octet
group of 8 bits

3.9
organizationally unique identifier
OUI
globally unique 24-bit value assigned to a company or other organisation for use in MAC addresses and other identifiers, see Clause 9 of IEEE 802

3.10
point-to-multipoint call
unidirectional path across a network which conveys information from a single source unit to one or more destination units, the information being copied by the network wherever the path branches

3.11
point-to-point call
bidirectional path across a network which conveys information in both directions between two network interface units

3.12
port
external input to, or output from, a unit

3.13
unit
single piece of equipment

3.14 Other abbreviations

3.14.1
Audio Engineering Society
AES

3.14.2
abstract syntax notation one
ASN.1

3.14.3
basic encoding rules
BER

3.14.4
digital audio broadcasting
DAB

3.14.5
Internet Assigned Numbers Authority
IANA

3.14.6
Internet Engineering Task Force
IETF

3.14.7
Internet protocol
IP

3.14.8
International Telecommunications Union
ITU

3.14.9
ITU Telecommunication Standardization Sector
ITU-T

3.14.10
local area network
LAN

3.14.11
media access control
MAC

3.14.12
management information base
MIB

3.14.13
Motion Picture Experts Group
MPEG

3.14.14
radio data system
RDS

3.14.15
simple network management protocol
SNMP

3.14.16
Universal Resource Locator
URL

3.14.17
Unicode transformation format 8 bit encoding form
UTF-8

4 Unit management

4.1 Protocol

The protocol used for unit management shall be the SNMP, either version 1 as specified in IETF RFC1157 or version 2 as specified in IETF RFC1441 or version 3 as specified in IETF RFC3411-3418.

If SNMP version 1 is used, references in this standard to the RESPONSE message shall be construed as references to the GET RESPONSE message.

Each message shall be encapsulated as a service data unit for the packet transfer service provided by the network on which it is transmitted.

NOTE 1 Details of the packet transfer service are specified in the subpart of IEC 62379-5 or IEC 62379-6 which applies to the network or link which provides it.

NOTE 2 RFC1157 specifies that all units must support messages of up to 484 octets in length; if the packet transfer service includes negotiation of maximum message length, longer messages may be supported.

A managed unit shall support at least one version of SNMP; a management terminal shall support all versions.

NOTE 3 The management terminal may establish which version the managed unit supports by signalling during call set-up or simply by trial and error.

NOTE 4 If authentication is required, version 3 should be used (rather than using community names in v1 or v2). Similarly, if encryption is required.

4.2 MIB definitions

4.2.1 General

All network-managed control functions in conformant equipment shall be represented as instances of managed object types defined in this standard or other parts of this standard or elsewhere.

NOTE 1 The use of managed object types not defined by IEC 62379 is permitted to allow control of functions outside the scope of IEC 62379 using the standard protocol. Object types defined by IEC 62379 should be used where applicable, in preference to object types defined elsewhere.

NOTE 2 Managed objects are typically organized in groups of related functions.

Each managed object type in this or other parts of IEC 62379 is defined by the following attributes.

- *Identifier* specifies the name and number that identify the object relative to the group or object from which it descends.
- *Syntax* specifies the syntax of the abstract data structure representing the object value. The syntax is described using abstract syntax notation 1 (ASN.1), as specified in ISO/IEC 8824-1:1998.
- *Index* specifies whether the object is used to uniquely identify a row in a managed object table.
- *Readable* specifies the privilege level (as defined below) for read access to the object. A management call with a privilege level greater than, or equal to, this level shall be permitted to perform a get or get-next operation on the object. A read access level of *none* specifies that get and get-next operations are not permitted on the object.
- *Writable* specifies the privilege level (as defined below) for write access to the object. A management call with a privilege level greater than, or equal to, this level shall be permitted to perform a set operation on the object. A write access level of *none* specifies that set operations are not permitted on the object.
- *Volatile* specifies whether the current value of the object is retained after a hard reset or period of power loss:
  - *no* indicates that the value of the object is either built in to the unit or stored in non-volatile memory;
  - *yes* indicates that the value of the object is transient;
  - *maybe* indicates that the value of the object may or may not be volatile, depending on the design of the unit or on the value of some other object in the MIB.
- *Status* specifies the required level of implementation support for the object:
  - *mandatory* indicates that the object shall be implemented by all conformant equipment that also implements that object's parent group or object;
  - *optional* indicates that the object may or may not be implemented by any conformant equipment that also implements that object's parent group or object.
  - *deprecated* indicates that the object shall not be implemented by any newly designed conformant equipment.

For brevity, the status attributes are abbreviated to *m*, *o*, and *d* in object definition tables.

The privilege levels used for the *readable* and *writable* attributes are, from lowest to highest:

- *listener*
- *operator*
- *supervisor*
- *maintenance*

- *none*

NOTE 3 The managed object types are defined in this standard in a relatively concise, human-readable form. Each part of the standard also provides a machine-readable version of the definitions contained therein as an informative appendix. The machine-readable definitions are described using the structure for management information version 2 (SMIv2) as specified in IETF STD 58. Note that SMIv2 can only represent a subset of the above attributes.

4.2.2 Application-wide type definitions

The following application-wide types are used to specify the syntax of managed objects both in this standard and other parts of it.

NOTE 1 The following three types are used to ensure compatibility with SMIv2, which requires all integer values to be representable in 32 bits and places further restrictions on integer values used to index tables. They are not meant to imply that equipment must accept or store the full range of values.

```
IntegerNumber::= INTEGER (-2147483648..2147483647)
-- An unrestricted integer value.

CardinalNumber::= INTEGER (0..2147483647 )
-- A zero or positive integer value.

IndexNumber::= INTEGER (1..2147483647)
-- A positive integer value.
```

NOTE 2 An `IndexNumber` is thus an integer that may be used to index a table. This type is used where the numbering is consecutive from 1 upwards. In other cases a specific "handle" type, such as `ClockSource` or `SoftwareArea`, is defined.

NOTE 3 The following two types are also used to ensure compatibility with SMIv2, which does not permit use of the standard ASN.1 types `BOOLEAN` and `UTF8String`. The `TruthValue` type maps directly onto the type of the same name defined in SMIv2.

```
TruthValue::= INTEGER {
  true  (1),
  false (2)
} (true..false)
-- An enumeration representing a boolean value.

Utf8String::= OCTET STRING (SIZE(0..80))
-- A UTF-8 encoded text string.

Octet::= OCTET STRING (SIZE(1))
-- An arbitrary value that can be represented in 8 bits.

BlockId::= INTEGER (1..2147483647)
-- A handle uniquely identifying a control block within the unit.

BlockType::= OBJECT IDENTIFIER
-- An object identifier identifying a defined control block. The block
-- may be one defined in any Part of IEC 62379 or one defined elsewhere.

MediaFormat::= OBJECT IDENTIFIER
-- An object identifier identifying a defined media format. The format
-- may be one defined in any Part of IEC 62379 or one defined elsewhere.

PortDirection::= INTEGER {
  input (1),
  output (2)
} (input..output)
-- An enumeration identifying whether a port is an input or an output.

StatusSource::= INTEGER {
  unitWide (0)
```

```
} (unitWide | BlockId)
-- A handle uniquely defining a status broadcast source within the unit.

SourceBlockId::= INTEGER {
  nullBlock  (0)
} (nullBlock | BlockId)
-- A handle uniquely identifying a control block within the unit which
-- can also take a null value.

PrivilegeLevel::= INTEGER {
  listener    (1),
  operator    (2),
  supervisor  (3),
  maintenance (4)
} (listener..maintenance)
-- An enumeration identifying a call privilege level.
```

The following application-wide types are used to specify the syntax of managed objects defined in this standard.

```
MacAddress::= OCTET STRING (SIZE(6))
-- A 48-bit address from the number-space used for IEEE802 MAC addresses.
-- See 9.2 of IEEE Std 802-2001

TDomain::= OBJECT IDENTIFIER
-- An object identifier identifying a defined network address type. The
-- address type may be one defined in any Part of IEC 62379 or one
-- defined elsewhere.

TAddress::= OCTET STRING (SIZE(1..255))
-- A network specific address. Semantics should be explicitly defined
-- by an associated MIB object but may also be implicitly defined by
-- the capabilities of the equipment.

UnitIdentity::= OCTET STRING (SIZE(10))
-- A code value that uniquely identifies the equipment type
--   octets 0..2 = oui
--   octets 3..4 = manufacturerId
--   octets 5..8 = productId
--   octet  9    = modLevel

ResetCause::= INTEGER {
  unknown  (1),
  powerUp  (2),
  managed  (3),
  watchdog (4)
} (unknown..watchdog)
-- An enumeration identifying the cause of the last unit reset.

ResetType::= INTEGER {
  hard     (1),
  soft     (2),
  shutdown (3)
} (hard..shutdown)
-- An enumeration identifying a reset or shutdown operation.

PowerType::= INTEGER {
  ac     (1), -- external AC power supply
  dc     (2), -- external DC power supply
  stored (3)   -- internal battery or other charge storage device
} (ac..stored)
-- An enumeration identifying a power source type.
```

```
PowerStatus::= INTEGER {
  charged     (1),
  charging    (2),
  discharging (3),
  discharged  (4),
  faulty      (5),
  expired     (6)
} (charged..expired)
-- An enumeration identifying the status of a power source.

ChargeLevel::= INTEGER (0..100)
-- A value representing the charge level of a battery or other stored
-- charge device as a percentage.

UnitAlarms::= OCTET STRING (SIZE(1))
-- a set of bits representing the general alarms for a unit
--   bit 1 (lsb) = current power source alarm
--   bit 2       = other   power source alarm
--   bit 3       = time server 1 alarm
--   bit 4       = time server 2 alarm
--   bit 5       = reference clock alarm
--   bit 6       = reserved
--   bit 7       = reserved
--   bit 8 (msb) = reserved

DateTime::= OCTET STRING (SIZE(9))
-- A code value representing a date and time
--   octets 0..1 = year
--   octet  2    = month       (1..12)
--   octet  3    = day         (1..31)
--   octet  4    = hour        (0..23)
--   octet  5    = minute      (0..59)
--   octet  6    = second      (0..60)   (allows leap seconds)
--   octets 7..8 = millisecond (0..999)

ServerNumber::= INTEGER (1..2)
-- A value identifying one of two possible servers used for synchronising
-- an internal real time clock.

ServerType::= INTEGER {
  none    (1),
  private (2),
  network (3)
} (none..network)
-- An enumeration identifying the type of server being used for
-- synchronising an internal real time clock.

ClockSource::= INTEGER (1..255)
-- A handle uniquely identifying a reference clock source for a unit.
-- Semantics are equipment specific.

SoftwareArea::= INTEGER (1..2147483647)
-- A handle uniquely identifying a software upload area within the unit.

SoftwareClass::= INTEGER {
  general  (1),
  firmware (2)
} (1..255)
-- An enumeration identifying the storage class of a software upload
-- area. Semantics are equipment specific.
```

```
SoftwareAccess::= INTEGER {
  read  (1),
  write (2),
  erase (3),
  full  (4)
} (read..full)
-- An enumeration identifying the access permissions for a software
-- upload area.

SoftwareStatus::= INTEGER {
  empty        (1),
  erasing      (2),
  invalid      (3),
  writing      (4),
  beingWritten (5),
  valid        (6)
} (empty..valid)
-- An enumeration identifying the current state of a software upload
-- area.

SoftwareType::= INTEGER (1..2147483647)
-- A handle uniquely identifying a software content type. Semantics are
-- equipment specific.

SoftwareSerial::= INTEGER {
  none (16)
} (0..15 | none)
-- A handle used to identify the most recently uploaded content for a
-- software upload area.

SoftwareVersion::= OCTET STRING (SIZE(0..80))
-- A code value identifying the version of software contained in a
-- software upload area.

SoftwareLength::= INTEGER (1..256)
-- A value representing the length of a sequence of software content
-- within a software upload area.

SoftwareContents::= OCTET STRING (SIZE(0..256))
-- A sequence of software content within a software upload area.

OperationId::= OCTET STRING (SIZE(10))
-- A handle uniquely identifying a scheduled operation within the unit.

OperationType::= INTEGER {
  modify  (1),
  monitor (2)
} (modify..monitor)
-- An enumeration representing the action performed by a scheduled
-- operation.

ObjectName::= OBJECT IDENTIFIER
-- An object identifier identifying any MIB object implemented by the
-- unit.

ObjectValue::= OCTET STRING (SIZE(1..65535))
-- A code value representing the value of any MIB object implemented by
-- the unit. Coded using the ASN.1 Basic Encoding Rules (BER).

OperationState::= INTEGER {
  pending (1),
  delayed (2),
  aborted (3)
```

```
} (pending..aborted)
-- An enumeration representing the current state of a scheduled
-- operation.
```

4.2.3 Conceptual row type definitions

The following types are used to specify the syntax of managed objects in this standard that represent conceptual table rows.

```
UnitPowerSourceEntry::= SEQUENCE {
  psNumber       IndexNumber,
  psType         PowerType,
  psStatus       PowerStatus,
  psChargeLevel  ChargeLevel,
  psChargeTime   CardinalNumber
}

BlockEntry::= SEQUENCE {
  blockId    BlockId,
  blockType  BlockType
}

ConnectorEntry::= SEQUENCE {
  connRxBlockId      BlockId,
  connRxBlockInput   IndexNumber,
  connTxBlockId      BlockId,
  connTxBlockOutput  IndexNumber
}

ModeEntry::= SEQUENCE {
  mBlockId       BlockId,
  mBlockOutput   IndexNumber,
  mMediaFormat   MediaFormat,
  mEnabled       TruthValue
}

TimeServerEntry::= SEQUENCE {
  tsNumber       ServerNumber,
  tsType         ServerType,
  tsAddressType  TDomain,
  tsAddressValue TAddress,
  tsConnectTime  CardinalNumber,
  tsLockedTime   CardinalNumber,
  tsDifference   IntegerNumber
}

ClockOutputEntry::= SEQUENCE {
  coNumber     IndexNumber,
  coName       Utf8String,
  coFrequency  CardinalNumber
}

SoftwareAreaEntry::= SEQUENCE {
  swaId       SoftwareArea,
  swaClass    SoftwareClass,
  swaAccess   SoftwareAccess,
  swaStatus   SoftwareStatus,
  swaLength   CardinalNumber,
  swaType     SoftwareType,
  swaSerial   SoftwareSerial,
  swaVersion  SoftwareVersion
}
```

```
  SoftwareContentEntry::= SEQUENCE {
    swcAreaId  SoftwareArea,
    swcOffset  CardinalNumber,
    swcLength  SoftwareLength,
    swcData    SoftwareContents
  }

  ScheduledOpEntry::= SEQUENCE {
    soRequestId    OperationId,
    soRelativeId   OperationId,
    soType         OperationType,
    soPersistent   TruthValue,
    soDateTime     DateTime,
    soObjectName   ObjectName,
    soObjectValue  ObjectValue,
    soDescription  Utf8String,
    soPrivilege    PrivilegeLevel,
    soState        OperationState
}
```

4.2.4 MIB objects for basic unit identity and status information

The group of objects in Table 1 shall be implemented by all compliant equipment. The root node for these objects shall be:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) unit-information(1) }

Table 1 – Managed objects conveying information about the unit

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
unitName(1)	Utf8String		listener	supervisor	no	m
unitLocation(2)	Utf8String		listener	supervisor	no	m
unitAddress(3)	MacAddress		listener	none	no	m
unitIdentity(4)	UnitIdentity		listener	none	no	m
unitManufacturerName(5)	Utf8String		listener	none	no	m
unitProductName(6)	Utf8String		listener	none	no	m
unitSerialNumber(7)	Utf8String		listener	none	no	m
unitFirmwareVersion(8)	Utf8String		listener	none	no	o
unitUpTime(9)	CardinalNumber		listener	none	yes	m
unitResetCause(10)	ResetCause		listener	none	no	m
unitReset(11)	ResetType		none	supervisor	yes	o
unitPowerSource(12)	IndexNumber		listener	supervisor	maybe	m
unitPowerSourceTable(13)	SEQUENCE OF UnitPowerSourceEntry		none	none	no	m
└unitPowerSourceEntry(1)	UnitPowerSourceEntry		none	none	no	m
├psNumber(1)	IndexNumber	yes	none	none	no	m
├psType(2)	PowerType		listener	none	no	m
├psStatus(3)	PowerStatus		listener	none	yes	m
├psChargeLevel(4)	ChargeLevel		listener	none	yes	o
└psChargeTime(5)	CardinalNumber		listener	none	yes	o
unitAlarmsEnabled(14)	UnitAlarms		listener	supervisor	no	o
unitAlarmsRaised(15)	UnitAlarms		listener	none	yes	o

4.2.4.1 unitName

The name assigned to the unit. This is an arbitrary text string assigned by the system manager.

NOTE If a unit also implements the MIB object 1.3.6.1.2.1.1.5.0 (sysName) defined in RFC1213, unitName should be an alias for that object.

4.2.4.2 unitLocation

The physical location of the unit. This is an arbitrary text string assigned by the system manager.

NOTE If a unit also implements the MIB object 1.3.6.1.2.1.1.6.0 (`sysLocation`) defined in RFC1213, `unitLocation` should be an alias for that object.

4.2.4.3 unitAddress

The 48-bit address of the unit. This shall be an individual LAN MAC address as specified by 9.2 of IEEE Std 802 (not a multicast address) and should be the globally unique address of one of its interfaces. However, if a unit does not have a globally unique address a local address, which shall be unique within the network, shall be used. How this local address is allocated is outside the scope of this standard.

4.2.4.4 unitIdentity

A collection of information that uniquely identifies the product of which the unit is an example.

- `oui` is the OUI used by the product manufacturer for this product;
- `manufacturerId` is the unique identifier assigned to the manufacturer by the owner of the OUI;
- `productId` is the unique identifier assigned to the product by the manufacturer; and
- `modLevel` identifies any modifications made to the design of the product, or to the unit after it was manufactured.

4.2.4.5 unitManufacturerName

The name of the unit's manufacturer. This is an arbitrary text string assigned by the unit's manufacturer.

4.2.4.6 unitProductName

The product name assigned to the unit by its manufacturer. This is an arbitrary text string assigned by the unit's manufacturer.

4.2.4.7 unitSerialNumber

The serial number assigned to the unit by its manufacturer. This is an arbitrary text string assigned by the unit's manufacturer.

4.2.4.8 unitFirmwareVersion

The version number of the firmware installed in the unit. This is an arbitrary text string assigned by the unit's manufacturer.

This object shall be mandatory if a piece of equipment does not support firmware update via the group of objects specified in 4.2.8.

4.2.4.9 unitUpTime

The number of seconds since the unit last powered up or was reset.

4.2.4.10 unitResetCause

The cause of the last unit reset.

4.2.4.11 unitReset

Causes the unit to reset or shutdown (as specified by the set value). After a reset or shutdown initiated by writing to this object, the reset cause shall be set to `managed`.

4.2.4.12 unitPowerSource

The index number of the entry in the unit power source table that represents the current source of power for the unit. Only writable if the unit permits manual switching between power sources.

4.2.4.13 unitPowerSourceTable

A table of power source descriptors. Each power source in the unit has an entry in this table.

4.2.4.14 unitPowerSourceEntry

An entry in the unit power source table.

4.2.4.15 psNumber

The power source number. Used as an index when accessing the unit power source table. Each power source in a unit has a unique number.

NOTE Power source numbers should be allocated sequentially, starting from 1.

4.2.4.16 psType

The power source type.

4.2.4.17 psStatus

The current status of this power source:

- `charged` indicates the power source is fully charged. For an external supply this is used to indicate that the supply voltage is above the threshold required by the unit.
- `charging` indicates that the power source is a battery (or other stored charge device) that is recharging itself from another power source but is not yet fully charged.
- `discharging` indicates that the power source is a battery (or other stored charge device) that is not recharging itself from another power source and is not fully charged or fully discharged.
- `discharged` indicates that the power source is fully discharged. For an external supply this is used to indicate that the supply voltage is below the threshold required by the unit.
- `faulty` indicates that a fault has been detected with the power source.
- `expired` indicates that the power source is a battery (or other stored charge device) which is due for replacement.

4.2.4.18 psChargeLevel

The current charge level of this power source as a percentage of full charge.

4.2.4.19 psChargeTime

If the current status of this power source is `charging`, this indicates the estimated time (in minutes) until the power source is fully charged. If the current status of this power source is `discharging`, this indicates the estimated time (in minutes) until the power source is fully discharged. If the current status of this power source is any other value, this has the value `0`.

4.2.4.20 unitAlarmsEnabled

Indicates which alarms are enabled for this unit. Bits representing alarms that are not implemented by the unit shall always be read as zero.

4.2.4.21 unitAlarmsRaised

Indicates which alarms are raised for this unit.

4.2.5 MIB objects for the block framework

The group of objects in Table 2 shall be implemented by all compliant equipment. The root node for these objects shall be

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) block-framework(2) }

Table 2 – Managed objects for block and connector configuration

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
blockTable(1)	SEQUENCE OF BlockEntry		none	none	no	m
└blockEntry(1)	BlockEntry		none	none	no	m
├blockId(1)	BlockId	yes	none	none	no	m
└blockType(2)	BlockType		listener	supervisor	no	m
connectorTable(2)	SEQUENCE OF ConnectorEntry		none	none	no	m
└connectorEntry(1)	ConnectorEntry		none	none	no	m
├connRxBlockId(1)	BlockId	yes	none	none	no	m
├connRxBlockInput(2)	IndexNumber	yes	none	none	no	m
├connTxBlockId(3)	SourceBlockId		listener	supervisor	no	m
└connTxBlockOutput(4)	IndexNumber		listener	supervisor	no	m
modeTable(3)	SEQUENCE OF ModeEntry		none	none	no	m
└modeEntry(1)	ModeEntry		none	none	no	m
├mBlockId(1)	BlockId	yes	none	none	no	m
├mBlockOuput(2)	IndexNumber	yes	none	none	no	m
├mMediaFormat(3)	MediaFormat	yes	none	none	no	m
└mEnabled(4)	TruthValue		listener	supervisor	no	m

4.2.5.1 blockTable

A table of block descriptors. Each block in the unit has an entry in this table.

4.2.5.2 blockEntry

An entry in the block table.

4.2.5.3 blockId

The block identifier. Used as an index when accessing the block table. Each block in a unit has a unique identifier independent of the block type.

4.2.5.4 blockType

The block type identifies the node in the object identifier tree that is the root for any managed objects used to control blocks of this type. Only writable if the unit allows blocks to be reconfigured. Writing the value NULL shall request deletion of the block. Writing a non-NULL value for a non-existent `blockId` shall request creation of a new block with all its inputs unconnected.

4.2.5.5 connectorTable

A table of connector descriptors. Each connector in the unit has an entry in this table.

4.2.5.6 connectorEntry

An entry in the connector table.

4.2.5.7 connRxBlockId

The block identifier of the block that is the receiving end of the connection. Used as an index when accessing the connector table.

4.2.5.8 connRxBlockInput

The input number of the block input that is the receiving end of the connection. Used as an index when accessing the connector table.

4.2.5.9 connTxBlockId

The block identifier of the block that is the transmitting end of the connection. Only writable if the unit allows connections to be reconfigured. The value `nullBlock` indicates that the input is not connected.

4.2.5.10 connTxBlockOutput

The output number of the block output that is the transmitting end of the connection. Only writable if the unit allows connections to be reconfigured.

4.2.5.11 modeTable

A table of mode descriptors. For each block output in the unit there will be one or more entries in this table, specifying the valid data formats for that output, and controlling whether the output is currently permitted to operate in that mode.

NOTE The mode table may be used for the following purposes:

- to allow generic controlling software to determine the capabilities of a particular unit;
- to allow manual control over the data format output by a particular block;
- to allow restrictions to be imposed on a block that automatically selects its output data format.

When used for manual control of the output data format, enabling one mode should automatically disable all other modes for that output.

4.2.5.12 modeEntry

An entry in the mode table.

4.2.5.13 mBlockId

The block identifier. Used as an index when accessing the mode table.

4.2.5.14 mBlockOutput

The block output number. Used as an index when accessing the mode table.

4.2.5.15 mMediaFormat

The media format associated with this mode. Used as an index when accessing the mode table.

4.2.5.16 mEnabled

If true, indicates that the associated block output may operate in this mode.

4.2.6 MIB objects for real time clock information

The group of objects in Table 3 shall be implemented by any unit containing an internal real time clock. The root node for these objects shall be

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) time(3) }

Table 3 – Managed objects for real time clock information

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
timeOfDay(1)	DateTime		listener	supervisor	no	m
timeZone(2)	INTEGER (-720..840)		listener	supervisor	no	m
timeAdvance(3)	INTEGER (0..120)		listener	supervisor	no	m
timeErrorThreshold(4)	INTEGER (0..250)		listener	supervisor	no	o
timeAlarmDelay(5)	INTEGER (0..240)		listener	supervisor	no	o
timeServerTable(6)	SEQUENCE OF TimeServerEntry		none	none	no	o
└timeServerEntry(1)	TimeServerEntry		none	none	no	m
├tsNumber(1)	ServerNumber	yes	none	none	no	m
├tsType(2)	ServerType		listener	supervisor	no	m
├tsAddressType(3)	TDomain		listener	supervisor	no	o
├tsAddressValue(4)	TAddress		listener	supervisor	no	o
├tsConnectTime(5)	CardinalNumber		listener	none	yes	m
├tsLockedTime(6)	CardinalNumber		listener	supervisor	yes	m
└tsDifference(7)	IntegerNumber		listener	none	yes	m

4.2.6.1 timeOfDay

The UTC date and time according to the unit's internal clock.

4.2.6.2 timeZone

The difference (in minutes) between local time and UTC time, excluding any adjustment made for daylight saving.

4.2.6.3 timeAdvance

The amount (in minutes) by which local time has been advanced for daylight saving.

4.2.6.4 timeErrorThreshold

The amount (in milliseconds) by which two time sources are allowed to differ before they are considered to be in disagreement.

4.2.6.5 timeAlarmDelay

The time (in seconds) for which a time source shall be in disagreement before an alarm is raised.

4.2.6.6 timeServerTable

The table of time server descriptors for this unit. There shall be two entries in this table.

4.2.6.7 timeServerEntry

An entry in the time server table.

4.2.6.8 tsNumber

The time server number. Used as an index when accessing the time server table.

4.2.6.9 tsType

The time server type.

4.2.6.10 tsAddressType

The type of network address used to locate and connect to this time server. This defines the semantics associated with `tsAddressValue`.

NOTE If the unit does not implement this object, the interpretation of `tsAddressValue` requires information conveyed other than by this standard, for instance, that the system only supports one kind of address. Equipment that only supports one kind of address type may implement this object as read-only, and it may be present even if the server type is not "network".

4.2.6.11 tsAddressValue

The network address used to locate and connect to this time server. Only used if the time server type is set to network. If the unit implements `tsAddressType`, the address value shall be interpreted according to the current value of `tsAddressType`.

4.2.6.12 tsConnectTime

The time (in seconds) that the unit has maintained an uninterrupted connection to this time server.

4.2.6.13 tsLockedTime

The time (in seconds) that the unit's internal clock has been in agreement with the time of day broadcast by this time server.

4.2.6.14 tsDifference

The difference (in milliseconds) between the unit's internal clock and the time of day broadcast by this time server. A positive difference indicates that the unit's clock is ahead, a negative difference indicates that it is behind.

4.2.7 MIB objects for reference clock information

The group of objects in Table 4 shall be implemented by any unit that uses or outputs a reference clock. The root node for these objects shall be:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) clock(4) }

Table 4 – Managed objects for reference clock information

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
clockSource(1)	ClockSource		listener	none	yes	m
clockFrequency(2)	CardinalNumber		listener	none	no	m
clockLockedTime(3)	CardinalNumber		listener	none	no	m
clockAlarmDelay(4)	CardinalNumber		listener	supervisor	no	o
clockOutputTable(5)	SEQUENCE OF ClockOutputEntry		none	none	no	o
└clockOutputEntry(1)	ClockOutputEntry		none	none	no	m
├coNumber(1)	IndexNumber	yes	none	none	no	m
├coName(2)	Utf8String		listener	supervisor	no	m
└coFrequency(3)	CardinalNumber		listener	supervisor	no	m

4.2.7.1 clockSource

The source of the reference clock signal.

4.2.7.2 clockFrequency

The nominal frequency (in Hz) of the reference clock signal. A value of zero indicates that no signal is present or that the frequency cannot be measured.

4.2.7.3 clockLockedTime

The time (in seconds) the unit has been locked to the reference clock signal.

4.2.7.4 clockAlarmDelay

The time (in seconds) for which the reference clock must be absent or out of range before an alarm is raised.

4.2.7.5 clockOutputTable

The table of clock output descriptors for this unit.

4.2.7.6 clockOutputEntry

An entry in the clock output table.

4.2.7.7 coNumber

The output number. Used as an index when accessing the clock output table.

4.2.7.8 coName

The name assigned to the clock output. This is an arbitrary text string assigned by the system manager.

4.2.7.9 coFrequency

The required nominal clock frequency (in Hz) for this output.

4.2.8 MIB objects for software upload

The group of objects in Table 5 shall be implemented by all equipment that supports remote uploading of software or other configuration files. The root node for these objects shall be:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) software(5) }

Table 5 – Managed objects for software upload

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
softwareAreaTable(1)	SEQUENCE OF SoftwareAreaEntry		none	none	no	m
└softwareAreaEntry(1)	SoftwareAreaEntry		none	none	no	m
├swaId(1)	SoftwareArea	yes	none	none	no	m
├swaClass(2)	SoftwareClass		maintenance	none	no	m
├swaAccess(3)	SoftwareAccess		maintenance	none	no	m
├swaStatus(4)	SoftwareStatus		maintenance	maintenance	no	m
├swaLength(5)	CardinalNumber		maintenance	maintenance	no	m
├swaType(6)	SoftwareType		maintenance	maintenance	no	m
├swaSerial(7)	SoftwareSerial		maintenance	maintenance	no	m
└swaVersion(8)	SoftwareVersion		maintenance	none	no	m
softwareContentTable(2)	SEQUENCE OF SoftwareContentEntry		none	none	no	m
└softwareContentEntry(1)	SoftwareContentEntry		none	none	no	m
├swcAreaId(1)	SoftwareArea	yes	none	none	no	m
├swcOffset(2)	CardinalNumber	yes	none	none	no	m
├swcLength(3)	SoftwareLength	yes	none	none	no	m
└swcData(4)	SoftwareContents		maintenance	maintenance	no	m

4.2.8.1 softwareAreaTable

The table of software upload area descriptors for this unit.

4.2.8.2 softwareAreaEntry

An entry in the software upload area table.

4.2.8.3 swaId

A handle uniquely identifying a software upload area within the unit. Used as an index when accessing the software upload area table.

4.2.8.4 swaClass

The storage class of this software upload area.

4.2.8.5 swaAccess

The accessibility of this software upload area.

4.2.8.6 swaStatus

The current status of this software upload area. A status of

- `empty` means the area has been erased (if applicable) and does not contain any data;
- `erasing` means a request to erase any data contained in the area has been accepted but the process has not yet been completed;
- `invalid` means the area is not empty but its format does not conform to any of the types supported by the unit;
- `writing` means the management terminal to which the RESPONSE is sent may alter the contents;
- `beingWritten` means another entity (another management terminal or the managed unit) may alter the contents; and
- `valid` means the area contains data in a format supported by the unit.

4.2.8.7 swaLength

The number of octets in this software upload area.

4.2.8.8 swaType

The type of software contained in this software upload area.

4.2.8.9 swaSerial

The serial number assigned to the last completed upload to this software upload area.

4.2.8.10 swaVersion

The coded version number for the contents of this software upload area. The version number is derived from the contents of the area in a way that depends on the type and is not defined here; it is zero-length if the contents do not include a version number.

4.2.8.11 softwareContentTable

The table of software upload content descriptors for this unit. This table has a notional entry for every possible contiguous sequence of octets within each software upload area.

4.2.8.12 softwareContentEntry

An entry in the software upload content table.

4.2.8.13 swcAreaId

A handle uniquely identifying a software upload area within the unit. Used as an index when accessing the software upload content table.

4.2.8.14 swcOffset

The offset (in octets) of this content sequence from the start of the specified software upload area. Used as an index when accessing the software upload content table.

4.2.8.15 swcLength

The length (in octets) of this content sequence. Used as an index when accessing the software upload content table.

4.2.8.16 swcData

The value of this content sequence.

4.2.9 MIB objects for scheduled operations

The group of objects in Table 6 shall be implemented by all equipment that supports scheduled operations. The root node for these objects shall be:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) schedule(6) }

Table 6 – Managed objects for scheduled operations

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
scheduledOpTable(1)	SEQUENCE OF ScheduledOpEntry		none	none	no	m
└scheduledOpEntry(1)	ScheduledOpEntry		none	none	no	m
├soRequestId(1)	OperationId	yes	none	none	no	m
├soRelativeId(2)	OperationId		listener	operator	no	m
├soType(3)	OperationType		listener	operator	no	m
├soPersistent(4)	TruthValue		listener	operator	no	m
├soDateTime(5)	DateTime		listener	operator	no	m
├soObjectName(6)	ObjectName		listener	operator	no	m
├soObjectValue(7)	ObjectValue		listener	operator	no	m
├soDescription(8)	Utf8String		listener	operator	no	m
├soPrivilege(9)	PrivilegeLevel		listener	operator	no	m
└soState(10)	OperationState		listener	operator	no	m

4.2.9.1 scheduledOpTable

The table of scheduled operation descriptors for this unit. The number of entries in the table is determined by the number of outstanding operations.

4.2.9.2 scheduledOpEntry

An entry in the table of scheduled operations.

4.2.9.3 soRequestId

The request ID for this operation. This is an arbitrary unique octet string value assigned by the managing unit making the request. Used as an index when accessing the scheduled operation list.

NOTE The request ID only needs to be unique with respect to other scheduled operations on the same unit.

4.2.9.4 soRelativeId

The ID of a previously scheduled operation that this operation must follow. If set to all zeroes, this operation is not required to follow any other operation.

NOTE An operation that is required to follow a previously scheduled operation is henceforth referred to as a relative operation, whilst an operation that is not required to follow a previously scheduled operation is referred to as an absolute operation.

4.2.9.5 soType

The type of this operation. This controls how the unit handles this operation.

NOTE An operation with a `soType` value of modify is henceforth referred to as a modify operation and an operation with a `soType` value of monitor is henceforth referred to as a monitor operation.

4.2.9.6 soPersistent

Whether this operation is persistent or transitory. If `soPersistent` is set to true, the operation remains in the table of scheduled operations after it has been executed, otherwise it is removed from the table of scheduled operations after it has been executed.

NOTE An operation with a `soPersistent` value of `true` is henceforth referred to as a persistent operation and an operation with a `soPersistent` value of `false` is henceforth referred to as a transitory operation.

4.2.9.7 soDateTime

For an absolute operation, this is the time at which the operation is scheduled to be executed. For a relative operation, this is the time at which the operation is scheduled to be executed relative to the actual time at which execution of the operation specified by `soRelativeId` was completed.

NOTE If an operation is delayed, any related operation is thus delayed by the same amount.

4.2.9.8 soObjectName

For a monitor operation, this is the MIB object instance to be monitored by this operation, otherwise this is the MIB object instance to be altered by this operation.

4.2.9.9 soObjectValue

For a monitor operation, this is the trigger value for the object instance specified by `soObjectName`; otherwise, it is the new value for the object instance specified by `soObjectName`.

NOTE The object value must be supplied as an octet string representing the ASN.1 BER encoded actual value. This is to ensure compatibility with SMIv2, which does not permit variant types for a managed object.

4.2.9.10 soDescription

An arbitrary text string describing this operation.

4.2.9.11 soPrivilege

The privilege level for this operation.

4.2.9.12 soState

The current state of this operation.

5 Procedures

5.1 Real-time clocks

Any unit containing an internal real-time clock that is not otherwise locked to an external reference clock shall attempt to continuously monitor the time of day broadcasts from two time servers connected to the network. When a connection is made to one of these servers, a local reference clock shall be created that is locked to the time broadcast by that server, with suitable filtering to reduce any jitter introduced by the network.

When a connection cannot be made to either time server, the unit's internal clock shall run free. When a connection can only be made to one server, the unit shall use the reference clock that is derived from that server to correct its internal clock. When a connection can be made to both servers, the unit shall check whether the reference clocks derived from the two servers are within the time error threshold set for the unit. If so, it shall use the mean of the two reference clocks to correct its internal clock. If not, it shall use the reference clock that is closest to its internal clock to correct its internal clock.

5.2 Procedures for uploading software

5.2.1 Areas

The management terminal shall request deletion of an area by setting its status to “erasing”; the managed unit shall then delete its current contents and set the status to “empty”. It may then amalgamate it with another empty area.

The management terminal shall request permission to write to an area by setting its status to “writing”; the managed unit shall reject the request if the previous status was anything other than “empty” or “valid”. If the request is successful, it will see the status as “writing” but all other management terminals will see it as “being written”. The type shall not be changed except after the status has been changed from “empty” to “writing” and before anything has been written to the contents (see 5.2.2). When all the contents have been written, the management terminal shall set the `status` to “valid”; it may be shown in the `RESPONSE` as “being written” if the managed unit needs time to write the type etc into the memory.

If the managed unit is reset, or the management connection is cleared down (in the case of a connection-oriented packet transfer service) or times out (in the case of a connectionless packet transfer service), while the status is “writing”, then if the previous status was “valid” it shall revert to “valid”, otherwise it shall change to “invalid”. The managed unit may erase invalid areas, converting them to “empty”.

Product-specific values for the `class` and `type` shall be defined by the manufacturer of the product; there should be just one set of type values for any given product, not one per class.

NOTE 1 The management terminal does not need to understand the significance of product-specific class or type values.

The length shall be the length of the area; the data it contains may be shorter.

NOTE 2 The method of indicating the length of the data is not defined here and is likely to be different for each type.

An area with serial number *s* (in range 0 to 15 inclusive) shall be a "latest" area of its type if there is no area of that type with serial number *s* + 1 (modulo 16).

If there is more than one "latest" area of a given type, or more than one area with the same type and serial number, or there are areas with all 16 possible serial number values for a given type, then the unit's memory is "malformed". A managed unit should reject any `SET` message that would cause its memory to become malformed. A management terminal that discovers a unit's memory is malformed should inform the user. The behaviour of a unit with

malformed memory, and the action to correct the problem, are product-specific and outside the scope of this standard.

If the unit's memory is not malformed, the "current" area of each type shall be selected as follows.

- Let *s* be the serial number of the latest area of the required type.
- The contents of the chosen area shall be checked; if the check fails then *s* shall be reduced by 1 (modulo 16) and if there is an area of the required type with serial number (the new value of) *s* this step shall be repeated.

NOTE 3 This standard does not specify what checks are to be carried out; they are likely to include calculating a checksum or CRC.

- If now there is an area of the required type with serial number *s*, it shall be designated the "current" area of that type; otherwise there is no "current" area of that type.

When loading software or other configuration information at start-up, the unit shall take it from the "current" area of the relevant type, which shall then be designated the "active" area of that type. When the "current" area changes (as a result of the procedure specified in 5.2.3), the unit may or may not change the "active" area to match.

NOTE 4 For software it will usually be appropriate for the "active" area to be the one chosen at start-up, but for some configuration information the unit might start using the new version immediately, without restarting. It will need to check the area first, to ensure it is "current" and not merely "latest".

An "active" area shall be read-only.

NOTE 5 This ensures that the existing software cannot be deleted until a new software has been loaded and is running.

5.2.2 Contents

A `SET` message for an entry in the "contents" table shall be treated as a request to write the contents of the area identified by the `handle`. The managed unit shall reply with the "readOnly" error code if the status of the area as seen by the sender of the `SET` message is not "writing".

NOTE 1 The length is limited by the size of octet string that will fit in a maximum-length message.

When the managed unit receives a `SET` message, it shall write the physical memory, then read it back and put the result in the `RESPONSE`. A `GET` or `SET` for a combination of offset and length which does not fit within the length of the area may be rejected (with the "tooBig" error code), or may be truncated, in which case the length in the `RESPONSE` shall reflect the length actually written.

NOTE 2 The `RESPONSE` serves as a confirmation not only that the message has been received but also that the data have been written correctly. The time required to write the memory and read it back is not likely to be long enough to delay the `RESPONSE` significantly. Performance when writing an area will be improved if the management terminal sends the second `SET` message without waiting for the response to the first, but if it sends too many `SET` messages before waiting for a response the managed unit (or the network) may lose some of them for lack of buffer space.

5.2.3 Procedure for updating a software component

Each component shall be associated with a class and type of area (see 5.2.5).

The procedure whereby a management terminal updates a component of a given type shall be as follows.

- The “areas” table shall first be scanned to locate an empty area of sufficient size and of the required class, and to find the serial number of the latest area of the required type. If there are no areas of the required class, it shall search for an empty area of class 1

instead. When searching for existing components, it should ignore the “class”. If the unit's memory is malformed, the process should be aborted and user intervention requested to delete the unwanted versions.

NOTE 1 If no suitable empty area is found, it may be possible to free up some space by erasing older versions of the component. If there are already 15 versions present, the oldest should be erased because otherwise the memory will become malformed when the 16th is written; the management terminal should not expect the managed unit to erase it on its own initiative.

- The status of the chosen area shall be set to “writing” and then the type and length of the area shall be set as required for the software component to be loaded. The length may be rounded up by the managed unit, for instance (in the case of flash memory), to be a whole number of blocks minus the space required to hold the type etc. The value returned in the `RESPONSE` shall be the actual length, after rounding up.
- The serial number may be written either before or after the data; its value shall be 1 more (modulo 16) than the serial number of the latest area of the same type; or any value in the range 0 to 15 if no area of that type was found.
- The management terminal shall then write the data to the area. It should use a window size of 2, i.e. at any time have up to 2 (but no more) `SET` messages for which no `RESPONSE` has yet been received. Thus it should start by sending two `SET` messages, then wait until it has seen the `RESPONSE` to the first before sending the third.
- Finally, the status shall be set to “valid”.

NOTE 2 A `unitReset` will normally be required before the unit will run the new software.

5.2.4 File format for a software component

Each software component shall be supplied in a file in the following format.

The file shall begin with two lines of text coded according to ISO/IEC 646 (7-bit characters with zero in the top bit), as follows:

`#62379.iec.ch::`*[format]*`:`*[domain]*`#`*[product+cpt]*`~`*[version]*
`length =` *[length]*`[other text]`

The first line shall be terminated by either a single carriage return character (code $0D_{16}$) or a carriage return, line feed pair (codes $0D_{16}$ $0A_{16}$); the second line shall be terminated by a single carriage return character (code $0D_{16}$). The variable parts are:

[format]	either `memoryimagebinary` or `memoryimagehex`
[domain]	a domain name owned by the manufacturer of the managed unit
[product+cpt]	text which identifies the product and component
[version]	text which identifies the version
[length]	the number of octets, in hex (digits A-F may be in upper or lower case)
[other text]	reserved for future use (for example, to add a checksum); should be empty when written and ignored when read; if not empty, the first character shall be a semicolon (‘;’, code $3B_{16}$)

The *[product+cpt]* shall consist of zero or more characters with codes in the range 20_{16} to $7E_{16}$ inclusive. Its format shall be specified by the owner of the *[domain]*.

The *[version]* shall consist of zero or more characters with codes in the range 20_{16} to $7D_{16}$ inclusive. Its format shall be specified by the owner of the *[domain]*.

NOTE 1 These provisions forbid non-printing characters other than space; the second also forbids the tilde ('~') character in the version, so that the *[product+cpt]* consists of everything between the first '#' and the last '~' on the line.

EXAMPLE: `#62379.iec.ch::memoryimagehex:ninetiles.com#RM003 logic~0.1.0 BETA 38`
`length = 28C44`

The remainder of the file shall consist of a sequence of octet values.

If the format is `memoryimagehex` each octet value shall consist of two hexadecimal digits (A to F may be either upper or lower case). White space characters (codes 00_{16} to 20_{16} inclusive) may occur between octet values, also before the first and after the last, but not between the two digits of an octet.

NOTE 2 In this format the entire file is text but its size is more than twice the size of the data to be uploaded.

If the format is `memoryimagebinary` each octet value shall be stored directly in one octet of the file, with the first octet value being in the octet after the carriage return that terminates the second line.

NOTE 3 In this format the size of the file is a minimum.

In either format, if the number of octets is not exactly equal to the "length" value, the management terminal shall not use the file, but shall report to the user that it is corrupt.

NOTE 4 The management terminal is not expected to do anything with the sequence of octet values other than upload it to an area in the managed unit.

5.2.5 Format for product files

Software components to be uploaded shall be accompanied by a text file, referred to as a "product file", which contains information about one or more product types.

The file shall contain one or more lines each consisting of zero or more characters coded according to ISO/IEC 646. Lines shall be separated by carriage return (code $0D_{16}$). Line feed (code $0A_{16}$) may follow the carriage return and shall be ignored if it does.

The first line shall be "`#62379.iec.ch::productfile:formatversion1`". Subsequent lines that begin with a '#', also empty lines, are "comment" and shall be ignored. (The first line shall not be empty.)

Apart from the first line and comment lines, each line shall consist of a keyword, optionally followed by the character '=', and a value. When interpreting a line, the keyword shall be everything before the first '=', and the value shall be everything after the first '='; in each case leading and trailing white space characters (codes 00_{16} to 20_{16} inclusive) shall be removed, and any internal sequence of white space characters shall be replaced by a single space. If there is no '=' on the line, the keyword shall be the whole line (after removing/replacing white space characters) and the value shall be empty. Keywords shall be case insensitive.

The information about each product shall begin with a line with the keyword "`product`" and a value consisting of four hex numbers separated by spaces, which are the four numbers that make up the `unitIdentity`. It shall include everything until the next line with the keyword "`product`", or until the end of the file if there are no more such lines.

There may be a line with the keyword "`description`", the value of which shall be a human-readable description of the product.

For each software component, there shall be a line with a keyword which consists of "`software`", a space, the area type in hex, another space, and the area class in hex. The

value shall be the “location” of the file containing the component of that type; the format of a “location” is different for the two distribution methods, see 5.2.6.

EXAMPLE: The line "`software 90 02 = 4-1-2-0.txt`" shows that software in the area with type 90 and class 02 (both in hex) should be as in the file `4-1-2-0.txt`. The keyword is "`software 90 02`" and the value is "`4-1-2-0.txt`". There may be any number of white space characters (or none) either side of the "=" and at each end of the line, and any sequence of white space characters (but at least one) either side of the "`90`".

NOTE 1 The file itself is as specified in 5.2.5 and does not include a type or class; the same component may be used in different places with different type values.

NOTE 2 By comparing the part following the last ‘~’ in the first line of each software component file with the `versionNumber` of the current area of the same type from the “areas” table, the management terminal can discover whether the managed unit has the correct software loaded. The interpretation of version numbers is product-specific, so the management terminal should not attempt to check whether the software being loaded is “older” than the existing software.

5.2.6 Software distribution

5.2.6.1 Packaged

The product file and software component files may be distributed as a package, for instance, on a CD or by e-mailing a ZIP file.

When distributed in this way, a “location” shall be a filename within the package.

NOTE When new software is available, a new package should be distributed.

5.2.6.2 On-line

The product file distributed with the equipment may be an “indirect” product file, which does not specify the software direct; in place of the lines with “`software`” keywords there shall be a single line with the keyword “`URL`” and value a URL for the “current” product file.

The "current" product file should also be indirect; and contain the URL of a product file for a specific software version.

When distributed in this way, a “location” shall be a URL.

When new software is available, it should be uploaded to a server and the “current” product file changed to point to it.

NOTE A management terminal can check for new software by periodically downloading the “current” product file and seeing whether it has changed. It follows that the “current” product file must be changed when the software changes; it is not sufficient to simply upload new component files in place of the old ones.

5.3 Scheduled operations

5.3.1 Requesting a scheduled operation

A managed unit shall create a new entry in its `scheduledOpTable` when a `set` operation is performed by a managing unit that meets the following conditions:

- one of the variable bindings refers to an object instance that has a type prefix of `scheduledOpEntry`;
- the index value used to identify the aforementioned object instance does not match an existing entry;
- the object value supplied for the aforementioned object instance is an acceptable value;
- the response to the `set` operation has an error status of `noError`.

Any object values in the new entry that are not supplied by other variable bindings in the same `set` operation shall be initialized by the managed unit as described below.

If a value is supplied for `soRelativeId`, any `OperationId` value that references a pending scheduled operation on the managed unit shall be considered acceptable. If a value is not supplied, the managed unit shall use a value of all zeroes.

If a value is supplied for `soType`, any valid `OperationType` value supported by the managed unit shall be considered acceptable. If a value is not supplied, the managed unit shall use the value `modify`.

If a value is supplied for `soPersistent`, any valid `TruthValue` value shall be considered acceptable. If a value is not supplied, the managed unit shall use the value `false`.

If a value is supplied for `soDateTime`, any valid `DateTime` value that does not require the scheduled operation to be executed within the next 2 min shall be considered acceptable. If a value is not supplied, the managed unit shall use a value of current time plus 5 min for absolute operations or a value of 5 min for relative operations.

If a value is supplied for `soObjectName`, any `ObjectName` value that refers to a MIB object instance that is implemented by the managed unit and that can be written (for modify operations) or read (for monitor operations) at the privilege level specified for `soPrivilege` may be considered acceptable. If a value is not supplied, the managed unit shall use a null value.

NOTE The unit may implement this facility in a general way that allows any object to be written by a modify operation or read by a monitor operation (subject to privilege etc. restrictions) or it may support only a specific set of scheduled operations. In the latter case, only those values of `soObjectName` appropriate to the operations that are supported will be acceptable.

If a value is supplied for `soObjectValue`, any `OCTET STRING` value that, when decoded using the ASN.1 BER, results in a valid value for the object instance specified by `soObjectName` shall be considered acceptable. If a value is not supplied, the managed unit shall use a null value.

If a value is supplied for `soDescription`, any valid `Utf8String` value shall be considered acceptable. If a value is not supplied, the managed unit shall use a null string.

If a value is supplied for `soPrivilege`, any privilege level less than, or equal to, the privilege level of the management call that requested the `set` operation shall be considered acceptable. If a value is not supplied, the managed unit shall use the privilege level of the management call that requested the `set` operation.

If a value is supplied for `soState`, only the value `pending` shall be considered acceptable. If a value is not supplied, the managed unit shall use the value `pending`.

A `set` operation that attempts to write to a non-existent entry in the `scheduledOpTable` but that does not meet all the above conditions shall be rejected by the managed unit.

Whilst the value of `soState` in an entry in a unit's `scheduledOpTable` is `pending` or `delayed`, any writeable object instance associated with that entry may be modified by subsequent `set` operations, provided that the privilege level of the management call making the modification is greater than, or equal to, the value of `soPrivilege` in that entry, and that the associated scheduled operation is not due to be executed within the next 2 min. The acceptable values for such object instances shall be as described above, except that any valid `OperationState` value shall be accepted for `soState`.

5.3.2 Executing a scheduled operation

Whenever the current value of `timeOfDay` is greater than, or equal to, the value of `soDateTime` for a given absolute operation, and the value of `soState` for that operation is `pending`, the operation shall be executed. Execution of a modify operation shall comprise of setting the object instance specified by `soObjectName` to the value specified by `soObjectValue`. Execution of a monitor operation shall comprise of waiting for the value of the object instance specified by `soObjectName` to be equal to the value specified by `soObjectValue`.

After an operation has been executed, any relative operation whose `soRelativeId` value references the executed operation shall be converted into an absolute operation by setting its `soRelativeId` value to all zeroes and adding the current value of `timeOfDay` to its `soDateTime` value. If a converted operation is a persistent operation the original values of `soRelativeId` and `soDateTime` shall be stored for future use.

For a transitory operation, the entry for the executed operation shall then be removed from the unit's `scheduledOpTable`. For a persistent operation, the original values of its `soRelativeId` and `soDateTime` shall be restored (if necessary) and, if it was originally an absolute operation, its `soState` shall be set to `delayed`.

5.3.3 Delaying a scheduled operation

When, as a result of a `set` operation, the value of `soState` in an existing entry in a unit's `scheduledOpTable` is set to `delayed`, the unit shall indefinitely delay execution of the associated scheduled operation. `soState` may only be modified if the privilege level of the management call making the modification is greater than or equal to the value of `soPrivilege` in that entry.

5.3.4 Aborting a scheduled operation

When, as a result of a `set` operation, the value of `soState` in an existing entry in the `scheduledOpTable` of a unit is set to `aborted`, the entry for the aborted operation shall be removed from the unit's `scheduledOpTable`.

5.3.5 State of relative operations

At any time that the `soRelativeId` of a relative operation is not equal to the `soRequestId` of any entry in the table of scheduled operations, the `soState` value for that operation shall be set to `delayed`.

6 Status broadcasts

6.1 General

A status broadcast shall be initiated by a unit in response to an appropriate management command or (if the network supports it) call set-up request.

NOTE 1 The process of initiating a status broadcast and the method of transporting the status information are network-specific and are described in the relevant subpart of IEC 62379-5 or IEC 62379-6.

Once a status broadcast has been initiated, the source unit shall transmit one or more "pages" of status information at regular intervals. The set of pages to be transmitted is a "page group" which shall be identified by a globally unique OID. Each page in the group shall be identified by a "page number" and the specification of a page group shall define the format and semantics associated with each page number.

The first two octets of each page shall contain the page number, which shall be coded as an unsigned integer value in the range 1 to 65535 inclusive (most significant octet first). The remaining octets shall be coded according to the page format. The maximum length of a page shall be 484 octets.

NOTE 2 The number is the same each time a page is broadcast, even if it contains different information. For instance, 6.3.3 specifies that page 4 reports all relevant operations in the list in rotation, so if there are several such operations it will report a different one each time the page is transmitted; however, in every case, the page number is coded as 4.

Integers shall be coded in binary using a fixed number of octets, with the first octet transmitted containing the most significant 8 bits of the value. Values are unsigned except where otherwise stated; signed numbers shall be coded using two's complement representation.

Text strings shall be coded using UTF-8 encoding, with the first octet transmitted being the first octet of the coded string.

A unit shall notionally contain one source of status information producing pages that relate to the entire unit, and, for each block in the unit for which status broadcasts are supported, one source of status information producing pages that relate to that block. The status source shall be identified when initiation of a status broadcast is requested.

Each page in a group may be broadcast at regular intervals or may be broadcast as required (but limited to a maximum rate). The broadcast rate may be different for each page in a group. Each page group shall have an associated "base page rate", and the specification of the page group shall relate the broadcast rate (or maximum rate) for each page in that group to the base page rate and shall define a default value for the base page rate.

The base page rate for a particular status broadcast call may be specified during call initiation by extending the OID that identifies the required page group by a single number that represents the base page rate in pages per minute. If a base page rate is not specified, the default base page rate shall be assumed. A unit may restrict the base page rate values it supports but shall as a minimum support the default base page rate for each page group it implements. The unit may broadcast at a different rate if the requested base page rate is not supported or the page is already being broadcast at a different rate.

A unit shall support simultaneous broadcast of all valid status source and page group combinations for the status sources and page groups it implements. It shall reject any request for a status broadcast with an invalid combination of status source and page group.

NOTE 3 A unit is only required to support a single broadcast of each page group from each status source. A request for a status broadcast of a page group that is already being broadcast from the requested source at a different base page rate may be rejected if the base page rate of the existing broadcast is not acceptable to the requester.

If the length of a received page is more than is implied by its definition, receiving equipment shall ignore the excess octets. If the length of a received page is less than is implied by its definition, receiving equipment shall fill the missing octets with zeroes.

6.2 Page formats

6.2.1 Basic unit identity page

This page shall be produced by the unit-wide status source (i.e. not by any of the blocks). It shall have the following content:

Octet(s)	*Description*	*Value*	*Note(s)*
1..2	Page number		
3..5	OUI	`unitIdentity.oui`	1
6..7	Manufacturer ID	`unitIdentity.manufacturerId`	1,2
8..11	Product ID	`unitIdentity.productId`	1,3
12	Modification level	`unitIdentity.modLevel`	1,3
13	Alarms raised	`unitAlarmsRaised`	1
14	Alarms enabled	`unitAlarmsEnabled`	1
15..20	MAC address	`unitAddress`	1
21..24	Up time	`unitUpTime`	1

NOTE 1 Coded with the current value of the specified object in the MIB (see 4.2.4).

NOTE 2 Assigned by the owner of the OUI.

NOTE 3 Assigned by the manufacturer.

6.2.2 Time-of-day page

This page shall be produced by the unit-wide status source. It shall have the following content.

Octet(s)	*Description*	*Value*	*Note(s)*
1..2	Page number		
3..4	Year	`timeOfDay.year`	1
5	Month	`timeOfDay.month`	1
6	Day	`timeOfDay.day`	1
7	Hour	`timeOfDay.hour`	1
8	Minute	`timeOfDay.minute`	1
9	Second	`timeOfDay.second`	1
10..11	Millisecond	`timeOfDay.millisecond`	1
12..13	Time zone	`timeZone`	1
14	Daylight saving	`timeAdvance`	1
15	Leap second sign	-1..1	2
16	Leap second month	0..12	3
17	Leap second zone	0..1	4
18..19	GPS week	0..65535	5,6,7
20..22	GPS second	0..604799	5,6
23	GPS to UTC offset	0..255	5,6

NOTE 1 Coded with the current value of the specified object in the MIB (see 4.2.6).

NOTE 2 Coded with 0 if no leap second is pending. Coded with 1 if a second is due to be inserted on the last day of the leap second month. Coded with -1 if a second is due to be skipped on the last day of the leap second month.

NOTE 3 Coded with 0 if no leap second is pending. Otherwise coded with the month number at the end of which a second is due to be inserted or skipped.

NOTE 4 Coded with 1 from 15 min before a leap second is due to occur until 15 min after a leap second has occurred. Coded with 0 at all other times.

NOTE 5 If the local time has been derived (directly or indirectly) from a GPS receiver, coded with the raw GPS data, otherwise coded with 0.

NOTE 6 Octets 18-23 may be omitted if local time is not derived from a GPS receiver.

NOTE 7 Implementers should note that GPS satellites only broadcast the least significant 10 bits of this value; from September 1999 to March 2019 the remaining bits are 000001.

6.2.3 Scheduled operations page

This page shall be produced by the unit-wide status source. It shall have the following content.

Octet(s)	*Description*	*Value*	*Note(s)*
1..2	Page number		
3..12	Request ID	`soRequestId`	1
13..21	Scheduled time	`soDateTime`	2, 4
22	Privilege level	`soPrivilege`	1
23	Status	`soState`	1
24	Description length	1..80	3
25..	Description	`soDescription`	1

NOTE 1 Coded with the current value of the specified object in the MIB from the list entry associated with the reported operation (see 4.2.9).

NOTE 2 For an absolute operation, coded with the current value of the `soDateTime` object in the MIB from the list entry associated with the reported operation. For a relative operation, coded with the sum of the current values of the `soDateTime` object in the MIB from the list entry associated with the reported operation and from the list entries found by tracing back through the current value of the `soRelativeId` object of each entry in turn until an entry for an absolute operation is reached. In either case, if the current value of the `soDateTime` object from the entry associated with the absolute operation is less than the current value of the `timeOfDay` object in the MIB, the latter value shall be used in place of the former.

NOTE 3 Coded with the number of octets required to code the following item.

NOTE 4 The scheduled time reported in this page is the latest estimate of when the reported operation will be executed, allowing for operations being delayed.

6.3 Page groups

6.3.1 `basicUnitStatus`

This group shall be supported by all units. It shall only be valid in combination with the unit-wide status source. A broadcast of this group shall consist of the following.

- page 1: Basic unit identity, broadcast at the base page rate

The default base page rate for this group shall be 60 pages per minute (1 page every second).

6.3.2 `timeOfDay`

This group may be supported by any unit that knows the time of day. It shall only be valid in combination with the unit-wide status source. A broadcast of this group shall consist of the following.

- page 2: Time of day, broadcast at the base page rate

The default base page rate for this group shall be 120 pages per minute (1 page every 500 ms). A call that carries this group shall be allocated bandwidth to support a peak rate of 1 page every 2 ms, to minimize latency and jitter.

6.3.3 `scheduledOps`

This group shall be supported by all units that support scheduled operations. It shall only be valid in combination with the unit-wide status source. A broadcast of this group shall consist of the following.

- page 3: Scheduled operation, broadcast at the base page rate, reporting the earliest pending scheduled operation which has a non-null description.
- page 4: Scheduled operation, broadcast at the base page rate, reporting each scheduled operation which has a non-null description and is not reported on page 3.

Pages 3 and 4 shall be broadcast alternately; page 4 shall report one scheduled operation on each occasion, taking all eligible operations (other than the one reported on page 3) in rotation.

The default base page rate for this group shall be 60 pages per minute (1 page of each number every second).

If at the time a page is to be transmitted there is no scheduled operation according to the above specification, the sender shall send a page containing only the two octets of the page number.

NOTE If the recipient appends zeroes as specified in 6.1, octet 23 will be zero which is not a valid `soState` value and may thus be used as an indication that there is no operation to report.

Annex A
(informative)

Background information

A.1 Relationship to IEC 62365

A.1.1 History

IEC 62365 was developed as AES47 to meet a requirement within radio broadcasting; it specifies how linear PCM audio is packed into ATM cells and how information about the audio format is conveyed when calls are set up.

It also includes a very simple message format sufficient to allow a management terminal to set up and clear down audio calls; this message format was intended as a temporary measure until a more comprehensive management protocol could be developed.

The ATM audio and video alliance (ATMAVA) then developed a common control interface for a trial of the use of AES47 in radio broadcasting.

IEC 62379 is based on this common control interface, the underlying principles of which are equally applicable to video, to other networking technologies, and to other applications in both professional and domestic environments.

A.1.2 Audio and video

IEC 62365 specifies connection of calls between audio “sources” and “destinations” without providing any detail as to what might lie between the termination point of the network circuit and the physical audio connector.

IEC 62379-2 provides (within the framework described in 0.3) a model of various processes that could be applied at each port, such as mixing audio from several sources (to fade across from one source to another, or for “conferencing” on speech circuits), individual gain controls for the sources, and format conversions such as between mono and stereo.

The model is intended to cover the most common signal processing functions that might be associated with an individual audio port; a given unit may implement as much or as little of this functionality as is required. If, for instance, an on/off function is provided in place of a gain control, the unit will only allow the gain to be set to 0 or 1.

IEC 62379-3 provides a similar model for video.

A.1.3 Status broadcasts

Status broadcasts are provided to allow status information to be supplied to any number of interested parties, repeated at regular intervals. Because these calls are implemented as point-to-multipoint connections, the maximum amount of work required to service status requests can be calculated irrespective of the number of interested parties. Three types of status broadcast call are defined in 6.3:

- unit status
- time of day
- scheduled operations

Unit status broadcasts are intended to contain all the information required for central monitoring equipment to detect faults or loss of power in the system. Time-of-day broadcasts allow accurate time-of-day information to be distributed to all units in the system. Other broadcasts, reporting information such as audio levels and calls that are connected across the network, are specified in other parts of IEC 62379.

The information is grouped into pages, each of which is formatted according to a specified page type. There is no constraint placed on the number of different pages that can be transmitted in a particular status call. Grouping the status information into pages makes it easier for a receiving unit to discard information in which it is not interested.

A.1.4 Data

Data calls are provided to allow non-audiovisual data to be transmitted point to point over the network. Three types of data call are defined in IEC 62379-4:

- logic state signalling data
- transparent asynchronous serial data
- time-critical data

Logic state signalling data calls provide transparent on/off switch connections across the network, which are intended to allow remote switching of equipment (such as the red light / buzz-back connections between studios).

Transparent asynchronous serial data calls convey streams of octets across the network, which can be used for remote control of equipment such as loudness processors. The physical port on the interface unit may be bit-serial (such as RS232 or RS422) or byte-serial (such as a parallel port on a PC). Incoming octets are collected by the source interface unit and packed into cells which are transmitted either when full or after a time-out.

Time-critical data calls are similar to transparent asynchronous serial data calls, but are used where the timing requirements are more severe; an example is control of video equipment which requires a message to be conveyed during the vertical blanking interval.

A data port is a source or destination of non-audiovisual data conveyed over the network. It can either be a physical port (for example, an external connector on the unit) or a logical port (for example, a defined point in the unit's internal data processing chain).

For certain applications (for example, red light connections) it may be desirable to combine (logically) the data from multiple switch data calls and feed the result to a single data output port. IEC 62379-4 specifies blocks which implement common data processing functions in a similar way to the audio and video processing blocks specified in IEC 62379-2 and IEC 62379-3.

A.2 Call reconnection

In a broadcast environment it is important that audio, video, or data calls that are part of the on-air programme chain be reconnected as soon as possible after a temporary power failure or a failure of part of the switch fabric. Similar requirements exist in other applications. IEC 62379 includes facilities for making the network interface units responsible for re-establishing lost connections. This could be regarded as an extension to encompass the session layer.

A network interface unit is required to reconnect a remembered destination that is disconnected for any reason other than because the unit was instructed via a management call to terminate the connection. It is also required to store the current remembered destinations in non-volatile memory, and to automatically re-establish these connections after

a temporary power failure. As this facility is only intended to cover short-term failures, a time limit is imposed, after which reconnection should not be attempted; this prevents a unit that is redeployed from one area to another attempting reconnection of calls that existed in its former location.

A.3 Call rerouting and replacement

At various times it may be necessary to remove a piece of equipment from the network, either because it has developed a fault or as part of scheduled maintenance work. It is desirable that this be done without disturbing any live streams being conveyed over the network that are currently routed through it. The following mechanism to achieve this is specified in the sub-part of IEC 62379-5 which applies to the network in question.

The first step is to instruct the unit that is about to be removed from the network not to accept any new connections. In case it is faulty and does not respond to this command, all the other equipment on the network can be instructed not to route any new connection through it.

Once this has been done, for each connection known to be routed through it, the relevant interface unit will be instructed to remake the connection.

A destination receiving an incoming connection that matches an existing connection will replace the call by accepting the new connection and then dropping the existing one. For audio and video calls it should perform a seamless changeover; in the case of audio it should use the sequence numbers specified in AES47 to align the old and new calls.

A.4 Call identity and importance

A desirable feature for the network switches used in a broadcast environment is that each switch knows the identity of the calls passing through it and whether or not they are part of an on-air programme chain. This allows a maintenance engineer to check that a particular switch is not carrying an important call before taking it out of service. Ideally, the switches would determine this information automatically.

This feature is also likely to be useful in other applications, for instance, in a home network to identify the source of a stream or to find whether anyone is watching the programme from a satellite or off-air receiver before changing channel.

Five items of information have been identified as being needed to characterize the identity and importance of a given call. These are:

- the name assigned to the input on the interface unit that is the source of the call;
- the name assigned to the output on the interface unit that is the destination of the call;
- the name assigned to the ultimate destination of the programme audio and/or video associated with the call;
- the importance assigned to the ultimate destination;
- the priority assigned to reconnecting the call if it is unexpectedly disconnected.

For brevity, these are referred to as the source name, the destination name, the service name, the importance, and the reconnection priority, and the whole group is referred to as the call identity. Note that a broadcast programme may be carried by a bundle of related calls, with separate calls for the metadata, for talkback, etc, and it is the ultimate destination of the programme audio which is of importance, not of the ancillary audio or data.

Because audio and video calls are point to multi-point connections, there will often be more than one destination for a given call. It is assumed that what is of interest is the call identity for the most important destination of the call.

Annex B
(informative)

Machine-readable MIB definitions

This annex provides a machine-readable version of the general MIB definitions which is intended to be interpretable by standard MIB browsing software tools. It does not express all the requirements of the standard, for instance, where access to an object is restricted at certain privilege levels. If there is any inconsistency between this annex and Clause 4, Clause 4 takes precedence.

The format used to describe the MIB objects conforms to IETF STD 58 (SMIv2).

```
IEC62379-1-MIB DEFINITIONS::= BEGIN

IMPORTS

  MODULE-IDENTITY, OBJECT-TYPE, Integer32
    FROM SNMPv2-SMI

  OBJECT-GROUP, MODULE-COMPLIANCE
    FROM SNMPv2-CONF

  TEXTUAL-CONVENTION, TruthValue, MacAddress, TDomain, TAddress
    FROM SNMPv2-TC;

generalMIB MODULE-IDENTITY
  LAST-UPDATED  "200701240000Z"
  ORGANIZATION  "IEC PT62379"
  CONTACT-INFO  "<not yet specified>"
  DESCRIPTION   "The MIB module for common control functions in IEC 62379
                 compliant equipment."
  REVISION      "200701240000Z"
  DESCRIPTION   "First CD."
 ::= { general 1 }

standard  OBJECT IDENTIFIER::= { iso 0 }

iec62379  OBJECT IDENTIFIER::= { standard 62379 }

general   OBJECT IDENTIFIER::= { iec62379 1 }

--
-- Object identifier values for module compliance statements
--

generalMIBCompliance OBJECT IDENTIFIER::= { generalMIB 0 }

--
-- Object identifier values for MIB object groups
--

unit      OBJECT IDENTIFIER::= { generalMIB 1 }
block     OBJECT IDENTIFIER::= { generalMIB 2 }
time      OBJECT IDENTIFIER::= { generalMIB 3 }
clock     OBJECT IDENTIFIER::= { generalMIB 4 }
software  OBJECT IDENTIFIER::= { generalMIB 5 }
schedule  OBJECT IDENTIFIER::= { generalMIB 6 }

--
-- Syntax definitions for MIB objects
--
```

```
IntegerNumber::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An integer value with no specific lower or upper limit. The
                 limits specified here are purely for SMIv2 compatibility."
  SYNTAX        Integer32

CardinalNumber::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A zero or positive integer value with no specific upper
                 limit. The upper limit specified here is purely for SMIv2
                 compatibility."
  SYNTAX        INTEGER (0..2147483647)

IndexNumber::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A positive integer value with no specific upper limit. The
                 upper limit specified here is purely for SMIv2 compatibility."
  SYNTAX        INTEGER (1..2147483647)

Utf8String::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A UTF-8 encoded text string."
  SYNTAX        OCTET STRING (SIZE(0..80))

BlockId::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A handle uniquely identifying a control block within a unit."
  SYNTAX        INTEGER (1..2147483647)

BlockType::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An object identifier identifying a defined control block.
                 The block may be one defined in any Part of IEC 62379 or
                 one defined elsewhere."
  SYNTAX        OBJECT IDENTIFIER

MediaFormat::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An object identifier identifying a defined media format. The
                 format may be one defined in any Part of IEC 62379 or one
                 defined elsewhere."
  SYNTAX        OBJECT IDENTIFIER

PortDirection::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration identifying whether a port is an input
                 or an output."
  SYNTAX        INTEGER {
                  input  (1),
                  output (2)
                } -- (input..output)

StatusSource::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A handle uniquely defining a status broadcast source within
                 the unit."
  SYNTAX        INTEGER (0..2147483647)
                -- {
                --   unitWide (0)
                -- } (unitWide | BlockId)

SourceBlockId::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A handle uniquely defining a control block within the unit
                 which can also take a null value."
  SYNTAX        INTEGER (0..2147483647)
                -- {
                --   nullBlock (0)
                -- } ( nullBlock  | BlockId)
```

```
PrivilegeLevel::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration identifying a call privilege level."
  SYNTAX        INTEGER {
                  listener    (1),
                  operator    (2),
                  supervisor  (3),
                  maintenance (4)
                } -- (listener..maintenance)

UnitIdentity::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A code value that uniquely identifies the equipment type:
                   octets 0..2 = oui
                   octets 3..4 = manufacturerId
                   octets 5..8 = productId
                   octet  9    = modLevel"
  SYNTAX        OCTET STRING (SIZE(10))

ResetCause::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration identifying the cause of the last unit reset."
  SYNTAX        INTEGER {
                  unknown  (1),
                  powerUp  (2),
                  managed  (3),
                  watchdog (4)
                } -- (unknown..watchdog)

ResetType::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration identifying a reset or shutdown operation."
  SYNTAX        INTEGER {
                  hard     (1),
                  soft     (2),
                  shutdown (3)
                } -- (hard..shutdown)

PowerType::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration identifying a power source type."
  SYNTAX        INTEGER {
                  ac     (1), -- external AC power supply
                  dc     (2), -- external DC power supply
                  stored (3)   -- internal battery or other charge storage device
                } -- (ac..stored)

PowerStatus::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration identifying the status of a power source."
  SYNTAX        INTEGER {
                  charged     (1),
                  charging    (2),
                  discharging (3),
                  discharged  (4),
                  faulty      (5),
                  expired     (6)
                } -- (charged..expired)
ChargeLevel::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A value representing the charge level of a battery or other
                 stored charge device as a percentage."
  SYNTAX        INTEGER (0..100)

UnitAlarms::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A set of bits representing the general alarms for a unit:
                   bit 1 (lsb) = current power source alarm
                   bit 2       = other   power source alarm
                   bit 3       = time server 1 alarm
```

```
                    bit 4       = time server 2 alarm
                    bit 5       = reference clock alarm
                    bit 6       = reserved
                    bit 7       = reserved
                    bit 8 (msb) = reserved"
    SYNTAX        OCTET STRING (SIZE(1))

  DateTime::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "A code value representing a date and time:
                    octets 0..1 = year
                    octet  2    = month       (1..12)
                    octet  3    = day         (1..31)
                    octet  4    = hour        (0..23)
                    octet  5    = minute      (0..59)
                    octet  6    = second      (0..60)  (allows leap seconds)
                    octets 7..8 = millisecond (0..999)"
    SYNTAX        OCTET STRING (SIZE(9))

  ServerNumber::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "A value identifying one of two possible servers used for
                   synchronising an internal real time clock."
    SYNTAX        INTEGER (1..2)

  ServerType::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "An enumeration identifying the type of server being used for
                   synchronising an internal real time clock."
    SYNTAX        INTEGER {
                    none    (1),
                    private (2),
                    network (3)
                  } -- (none..network)

  ClockSource::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "A handle uniquely identifying a reference clock source for a
                   unit. Semantics are equipment specific."
    SYNTAX        INTEGER (1..255)

  SoftwareArea::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "A handle uniquely identifying a software upload area within
                   the unit."
    SYNTAX        INTEGER (1..2147483647)

  SoftwareClass::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "An enumeration identifying the storage class of a software
                   upload area. Semantics are equipment specific."
    SYNTAX        INTEGER (1..255)
                  -- {
                  --   general  (1),
                  --   firmware (2)
                  -- } (general | firmware | 3..255)

  SoftwareAccess::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "An enumeration identifying the access permissions for a
                   software upload upload area."
    SYNTAX        INTEGER {
                    read  (1),
                    write (2),
                    erase (3),
                    full  (4)
                  } -- (read..full)

  SoftwareStatus::= TEXTUAL-CONVENTION
    STATUS        current
```

```
  DESCRIPTION   "An enumeration identifying the current state of a software
                 upload area."
  SYNTAX        INTEGER {
                  empty        (1),
                  erasing      (2),
                  invalid      (3),
                  writing      (4),
                  beingWritten (5),
                  valid        (6)
                } -- (empty..valid)

SoftwareType::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A handle uniquely identifying a software content type.
                 Semantics are equipment specific."
  SYNTAX        INTEGER (1..2147483647)

SoftwareSerial::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A handle used to identify the most recently uploaded content
                 for a software upload area."
  SYNTAX        INTEGER (0..16)
                -- {
                --   none (16)
                -- } (0..15 | none)

SoftwareVersion::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A code value identifying the version of software contained
                 in a software upload area."
  SYNTAX        OCTET STRING (SIZE(0..80))

SoftwareLength::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A value representing the length of a sequence of software
                 content within a software upload area."
  SYNTAX        INTEGER (1..256)

SoftwareContents::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A sequence of software content within a software upload area."
  SYNTAX        OCTET STRING (SIZE(0..256))

OperationId::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A handle uniquely identifying a scheduled operation within the
                 unit."
  SYNTAX        OCTET STRING (SIZE(10))

OperationType::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration representing the action performed by a scheduled
                 operation."
  SYNTAX        INTEGER {
                  modify  (1),
                  monitor (2)
                } -- (modify..monitror)

ObjectName::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An object identifier identifying any MIB object implemented by
                 the unit."
  SYNTAX        OBJECT IDENTIFIER

ObjectValue::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "A code value representing the value of any MIB object
                 implemented by the unit. Coded using the ASN.1 Basic
                 Encoding Rules (BER)."
  SYNTAX        OCTET STRING (SIZE(1..65535))
```

```
OperationState::= TEXTUAL-CONVENTION
  STATUS        current
  DESCRIPTION   "An enumeration representing the current state of a scheduled
                 operation."
  SYNTAX        INTEGER {
                  pending (1),
                  delayed (2),
                  aborted (3)
                } -- (pending..aborted)

--
-- MIB object definitions for basic unit identity and status information
--

unitName OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 1 }

unitLocation OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 2 }

unitAddress OBJECT-TYPE
  SYNTAX        MacAddress
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 3 }

unitIdentity OBJECT-TYPE
  SYNTAX        UnitIdentity
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 4 }

unitManufacturerName OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 5 }

unitProductName OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 6 }

unitSerialNumber OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 7 }

unitFirmwareVersion OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 8 }
```

```
unitUpTime OBJECT-TYPE
  SYNTAX        CardinalNumber
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 9 }

unitResetCause OBJECT-TYPE
  SYNTAX        ResetCause
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 10 }

unitReset OBJECT-TYPE
  SYNTAX        ResetType
  MAX-ACCESS    write-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 11 }

unitPowerSource OBJECT-TYPE
  SYNTAX        IndexNumber
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 12 }

UnitPowerSourceEntry::= SEQUENCE {
  psNumber       IndexNumber,
  psType         PowerType,
  psStatus       PowerStatus,
  psChargeLevel  ChargeLevel,
  psChargeTime   CardinalNumber
}

unitPowerSourceTable OBJECT-TYPE
  SYNTAX        SEQUENCE OF UnitPowerSourceEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unit 13 }

unitPowerSourceEntry OBJECT-TYPE
  SYNTAX        UnitPowerSourceEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
  INDEX         { psNumber }
 ::= { unitPowerSourceTable 1 }

psNumber OBJECT-TYPE
  SYNTAX        IndexNumber
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unitPowerSourceEntry 1 }

psType OBJECT-TYPE
  SYNTAX        PowerType
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.4."
 ::= { unitPowerSourceEntry 2 }

psStatus OBJECT-TYPE
  SYNTAX        PowerStatus
  MAX-ACCESS    read-only
  STATUS        current
```

```
    DESCRIPTION   "See IEC 62379-1 section 4.2.4."
   ::= { unitPowerSourceEntry 3 }

  psChargeLevel OBJECT-TYPE
    SYNTAX        ChargeLevel
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.4."
   ::= { unitPowerSourceEntry 4 }

  psChargeTime OBJECT-TYPE
    SYNTAX        CardinalNumber
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.4."
   ::= { unitPowerSourceEntry 5 }

  unitAlarmsEnabled OBJECT-TYPE
    SYNTAX        UnitAlarms
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.4."
   ::= { unit 14 }

  unitAlarmsRaised OBJECT-TYPE
    SYNTAX        UnitAlarms
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.4."
   ::= { unit 15 }

  unitGroup OBJECT-GROUP
    OBJECTS       { unitName,
                    unitLocation,
                    unitAddress,
                    unitIdentity,
                    unitManufacturerName,
                    unitProductName,
                    unitSerialNumber,
                    unitFirmwareVersion,
                    unitUpTime,
                    unitResetCause,
                    unitReset,
                    unitPowerSource,
                    psType,
                    psStatus,
                    psChargeLevel,
                    psChargeTime,
                    unitAlarmsEnabled,
                    unitAlarmsRaised }
    STATUS        current
    DESCRIPTION   "The group of objects that provide basic unit identity and
                   status information."
   ::= { unit 99 }

  --
  -- MIB object definitions for the block framework
  --

  BlockEntry::= SEQUENCE {
    blockId    BlockId,
    blockType  BlockType
  }

  blockTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF BlockEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.5."
   ::= { block 1 }
```

```
blockEntry OBJECT-TYPE
  SYNTAX        BlockEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
  INDEX         { blockId }
 ::= { blockTable 1 }

blockId OBJECT-TYPE
  SYNTAX        BlockId
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
 ::= { blockEntry 1 }

blockType OBJECT-TYPE
  SYNTAX        BlockType
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
 ::= { blockEntry 2 }

ConnectorEntry::= SEQUENCE {
  connRxBlockId      BlockId,
  connRxBlockInput   IndexNumber,
  connTxBlockId      SourceBlockId,
  connTxBlockOutput  IndexNumber
}

connectorTable OBJECT-TYPE
  SYNTAX        SEQUENCE OF ConnectorEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
 ::= { block 2 }

connectorEntry OBJECT-TYPE
  SYNTAX        ConnectorEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
  INDEX         { connRxBlockId, connRxBlockInput }
 ::= { connectorTable 1 }

connRxBlockId OBJECT-TYPE
  SYNTAX        BlockId
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
 ::= { connectorEntry 1 }

connRxBlockInput OBJECT-TYPE
  SYNTAX        IndexNumber
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
 ::= { connectorEntry 2 }

connTxBlockId OBJECT-TYPE
  SYNTAX        SourceBlockId
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.5."
 ::= { connectorEntry 3 }

connTxBlockOutput OBJECT-TYPE
  SYNTAX        IndexNumber
  MAX-ACCESS    read-write
  STATUS        current
```

```
    DESCRIPTION    "See IEC 62379-1 section 4.2.5."
   ::= { connectorEntry 4 }

  ModeEntry::= SEQUENCE {
    mBlockId       BlockId,
    mBlockOutput   IndexNumber,
    mMediaFormat   MediaFormat,
    mEnabled       TruthValue
  }

  modeTable OBJECT-TYPE
    SYNTAX         SEQUENCE OF ModeEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION    "See IEC 62379-2 section 4.4.1."
   ::= { block 3 }

  modeEntry OBJECT-TYPE
    SYNTAX         ModeEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION    "See IEC 62379-2 section 4.4.1."
    INDEX          { mBlockId, mBlockOutput, mMediaFormat }
   ::= { modeTable 1 }

  mBlockId OBJECT-TYPE
    SYNTAX         BlockId
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION    "See IEC 62379-2 section 4.4.1."
   ::= { modeEntry 1 }

  mBlockOutput OBJECT-TYPE
    SYNTAX         IndexNumber
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION    "See IEC 62379-2 section 4.4.1."
   ::= { modeEntry 2 }

  mMediaFormat OBJECT-TYPE
    SYNTAX         MediaFormat
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION    "See IEC 62379-2 section 4.4.1."
   ::= { modeEntry 3 }

  mEnabled OBJECT-TYPE
    SYNTAX         TruthValue
    MAX-ACCESS     read-write
    STATUS         current
    DESCRIPTION    "See IEC 62379-2 section 4.4.1."
   ::= { modeEntry 4 }


  blockGroup OBJECT-GROUP
    OBJECTS        { blockType,
                     connTxBlockId,
                     connTxBlockOutput,
                     mEnabled }
    STATUS         current
    DESCRIPTION    "The group of objects that identify or modify a unit's
                    functional block configuration."
   ::= { block 99 }

  --
  -- MIB object definitions for real-time clock information
  --

  timeOfDay OBJECT-TYPE
    SYNTAX         DateTime
```

```
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { time 1 }

timeZone OBJECT-TYPE
  SYNTAX        INTEGER (-720..840)
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { time 2 }

timeAdvance OBJECT-TYPE
  SYNTAX        INTEGER (0..120)
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { time 3 }

timeErrorThreshold OBJECT-TYPE
  SYNTAX        INTEGER (0..250)
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { time 4 }

timeAlarmDelay OBJECT-TYPE
  SYNTAX        INTEGER (0..240)
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { time 5 }

TimeServerEntry::= SEQUENCE {
  tsNumber        ServerNumber,
  tsType          ServerType,
  tsAddressType   TDomain,
  tsAddressValue  TAddress,
  tsConnectTime   CardinalNumber,
  tsLockedTime    CardinalNumber,
  tsDifference    IntegerNumber
}

timeServerTable OBJECT-TYPE
  SYNTAX        SEQUENCE OF TimeServerEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { time 6 }

timeServerEntry OBJECT-TYPE
  SYNTAX        TimeServerEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
  INDEX        { tsNumber }
 ::= { timeServerTable 1 }

tsNumber OBJECT-TYPE
  SYNTAX        ServerNumber
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
 ::= { timeServerEntry 1 }

tsType OBJECT-TYPE
  SYNTAX        ServerType
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.6."
```

```
    ::= { timeServerEntry 2 }

  tsAddressType OBJECT-TYPE
    SYNTAX        TDomain
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.6."
   ::= { timeServerEntry 3 }

  tsAddressValue OBJECT-TYPE
    SYNTAX        TAddress
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.6."
   ::= { timeServerEntry 4 }

  tsConnectTime OBJECT-TYPE
    SYNTAX        CardinalNumber
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.6."
   ::= { timeServerEntry 5 }

  tsLockedTime OBJECT-TYPE
    SYNTAX        CardinalNumber
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.6."
   ::= { timeServerEntry 6 }

  tsDifference OBJECT-TYPE
    SYNTAX        IntegerNumber
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.6."
   ::= { timeServerEntry 7 }

  timeGroup OBJECT-GROUP
    OBJECTS       { timeOfDay,
                    timeZone,
                    timeAdvance,
                    timeErrorThreshold,
                    timeAlarmDelay,
                    tsType,
                    tsAddressType,
                    tsAddressValue,
                    tsConnectTime,
                    tsLockedTime,
                    tsDifference }
    STATUS        current
    DESCRIPTION   "The group of objects used to control or monitor a unit's
                   internal real-time clock."
   ::= { time 99 }

  --
  -- MIB object definitions for reference clock information
  --

  clockSource OBJECT-TYPE
    SYNTAX        ClockSource
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.7."
   ::= { clock 1 }

  clockFrequency OBJECT-TYPE
    SYNTAX        CardinalNumber
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION   "See IEC 62379-1 section 4.2.7."
```

```
 ::= { clock 2 }

clockLockedTime OBJECT-TYPE
  SYNTAX        CardinalNumber
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
 ::= { clock 3 }

clockAlarmDelay OBJECT-TYPE
  SYNTAX        CardinalNumber
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
 ::= { clock 4 }

ClockOutputEntry::= SEQUENCE {
  coNumber     IndexNumber,
  coName       Utf8String,
  coFrequency  CardinalNumber
}

clockOutputTable OBJECT-TYPE
  SYNTAX        SEQUENCE OF ClockOutputEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
 ::= { clock 5 }

clockOutputEntry OBJECT-TYPE
  SYNTAX        ClockOutputEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
  INDEX         { coNumber }
 ::= { clockOutputTable 1 }

coNumber OBJECT-TYPE
  SYNTAX        IndexNumber
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
 ::= { clockOutputEntry 1 }

coName OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
 ::= { clockOutputEntry 2 }

coFrequency OBJECT-TYPE
  SYNTAX        CardinalNumber
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.7."
 ::= { clockOutputEntry 3 }

clockGroup OBJECT-GROUP
  OBJECTS       { clockSource,
                  clockFrequency,
                  clockLockedTime,
                  clockAlarmDelay,
                  coName,
                  coFrequency }
  STATUS        current
  DESCRIPTION   "The group of objects used to control or monitor reference
                 clocks used by or output by a unit."
 ::= { clock 99 }
```

```
--
-- MIB object definitions for software upload
--

SoftwareAreaEntry::= SEQUENCE {
  swaId       SoftwareArea,
  swaClass    SoftwareClass,
  swaAccess   SoftwareAccess,
  swaStatus   SoftwareStatus,
  swaLength   CardinalNumber,
  swaType     SoftwareType,
  swaSerial   SoftwareSerial,
  swaVersion  SoftwareVersion
}

softwareAreaTable OBJECT-TYPE
  SYNTAX        SEQUENCE OF SoftwareAreaEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { software 1 }

softwareAreaEntry OBJECT-TYPE
  SYNTAX        SoftwareAreaEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
  INDEX         { swaId }
 ::= { softwareAreaTable 1 }

swaId OBJECT-TYPE
  SYNTAX        SoftwareArea
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 1 }

swaClass OBJECT-TYPE
  SYNTAX        SoftwareClass
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 2 }

swaAccess OBJECT-TYPE
  SYNTAX        SoftwareAccess
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 3 }

swaStatus OBJECT-TYPE
  SYNTAX        SoftwareStatus
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 4 }

swaLength OBJECT-TYPE
  SYNTAX        CardinalNumber
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 5 }

swaType OBJECT-TYPE
  SYNTAX        SoftwareType
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
```

```
 ::= { softwareAreaEntry 6 }

swaSerial OBJECT-TYPE
  SYNTAX        SoftwareSerial
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 7 }

swaVersion OBJECT-TYPE
  SYNTAX        SoftwareVersion
  MAX-ACCESS    read-only
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareAreaEntry 8 }

SoftwareContentEntry::= SEQUENCE {
  swcAreaId  SoftwareArea,
  swcOffset  CardinalNumber,
  swcLength  SoftwareLength,
  swcData    SoftwareContents
}

softwareContentTable OBJECT-TYPE
  SYNTAX        SEQUENCE OF SoftwareContentEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { software 2 }

softwareContentEntry OBJECT-TYPE
  SYNTAX        SoftwareContentEntry
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
  INDEX         { swcAreaId, swcOffset, swcLength }
 ::= { softwareContentTable 1 }

swcAreaId OBJECT-TYPE
  SYNTAX        SoftwareArea
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareContentEntry 1 }

swcOffset OBJECT-TYPE
  SYNTAX        CardinalNumber
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareContentEntry 2 }

swcLength OBJECT-TYPE
  SYNTAX        SoftwareLength
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareContentEntry 3 }

swcData OBJECT-TYPE
  SYNTAX        SoftwareContents
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.8."
 ::= { softwareContentEntry 4 }

softwareGroup OBJECT-GROUP
  OBJECTS       { swaClass,
                  swaAccess,
                  swaStatus,
```

```
                    swaLength,
                    swaType,
                    swaSerial,
                    swaVersion,
                    swcData }
    STATUS          current
    DESCRIPTION     "The group of objects used to upload software or configuration
                     data to a unit."
   ::= { software 99 }

--
-- MIB object definitions for scheduled operations
--

ScheduledOpEntry::= SEQUENCE {
    soRequestId     OperationId,
    soRelativeId    OperationId,
    soType          OperationType,
    soPersistent    TruthValue,
    soDateTime      DateTime,
    soObjectName    ObjectName,
    soObjectValue   ObjectValue,
    soDescription   Utf8String,
    soPrivilege     PrivilegeLevel,
    soState         OperationState
}

scheduledOpTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF ScheduledOpEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "See IEC 62379-1 section 4.2.9."
   ::= { schedule 1 }

scheduledOpEntry OBJECT-TYPE
    SYNTAX          ScheduledOpEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "See IEC 62379-1 section 4.2.9."
    INDEX           { soRequestId }
   ::= { scheduledOpTable 1 }

soRequestId OBJECT-TYPE
    SYNTAX          OperationId
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "See IEC 62379-1 section 4.2.9."
   ::= { scheduledOpEntry 1 }

soRelativeId OBJECT-TYPE
    SYNTAX          OperationId
    MAX-ACCESS      read-write
    STATUS          current
    DESCRIPTION     "See IEC 62379-1 section 4.2.9."
   ::= { scheduledOpEntry 2 }

soType OBJECT-TYPE
    SYNTAX          OperationType
    MAX-ACCESS      read-write
    STATUS          current
    DESCRIPTION     "See IEC 62379-1 section 4.2.9."
   ::= { scheduledOpEntry 3 }

soPersistent OBJECT-TYPE
    SYNTAX          TruthValue
    MAX-ACCESS      read-write
    STATUS          current
    DESCRIPTION     "See IEC 62379-1 section 4.2.9."
   ::= { scheduledOpEntry 4 }
```

```
soDateTime OBJECT-TYPE
  SYNTAX        DateTime
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.9."
 ::= { scheduledOpEntry 5 }

soObjectName OBJECT-TYPE
  SYNTAX        ObjectName
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.9."
 ::= { scheduledOpEntry 6 }

soObjectValue OBJECT-TYPE
  SYNTAX        ObjectValue
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.9."
 ::= { scheduledOpEntry 7 }

soDescription OBJECT-TYPE
  SYNTAX        Utf8String
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.9."
 ::= { scheduledOpEntry 8 }

soPrivilege OBJECT-TYPE
  SYNTAX        PrivilegeLevel
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.9."
 ::= { scheduledOpEntry 9 }

soState OBJECT-TYPE
  SYNTAX        OperationState
  MAX-ACCESS    read-write
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 4.2.9."
 ::= { scheduledOpEntry 10 }

scheduleGroup OBJECT-GROUP
  OBJECTS       { soRelativeId,
                  soType,
                  soPersistent,
                  soDateTime,
                  soObjectName,
                  soObjectValue,
                  soDescription,
                  soPrivilege,
                  soState }
  STATUS        current
  DESCRIPTION   "The group of objects used to control or monitor a unit's
                 schedule of operations."
 ::= { schedule 99 }

--
-- Compliance statements
--

generalMIBComplianceV1 MODULE-COMPLIANCE
  STATUS        current
  DESCRIPTION   "The compliance statement for entities that conform to
                 IEC 62379-1 (200X)."
  MODULE
    MANDATORY-GROUPS  { unitGroup, blockGroup }

    GROUP       timeGroup
```

```
        DESCRIPTION "Mandatory for equipment that contains an internal real-time
                     clock."

        GROUP       clockGroup
        DESCRIPTION "Mandatory for equipment that uses or outputs a reference
                     clock."

        GROUP       softwareGroup
        DESCRIPTION "Mandatory for equipment that supports remote uploading of
                     software or configuration data."

        GROUP       scheduleGroup
        DESCRIPTION "Mandatory for equipment that supports scheduled operations."

      ::= {generalMIBCompliance 1 }

END
```

Annex C
(informative)

Machine-readable status page-group definitions

This annex provides a machine-readable version of the status page group definitions which is intended to be interpretable by standard MIB browsing software tools. If there is any inconsistency between this annex and Clause 6, Clause 6 takes precedence.

The format used to describe the status page group identifiers conforms to IETF STD 58 (SMIv2).

```
IEC62379-1-STATUS DEFINITIONS::= BEGIN

IMPORTS

  MODULE-IDENTITY, OBJECT-IDENTITY
    FROM SNMPv2-SMI

  iec62379
    FROM IEC62379-1-MIB;

generalStatusGroup MODULE-IDENTITY
  LAST-UPDATED  "200601080000Z"
  ORGANIZATION  "IEC PT62379"
  CONTACT-INFO  "<not yet specified>"
  DESCRIPTION   "The status page group identifiers defined in section 6.3 of
                 IEC 62379-1."
  REVISION      "200601080000Z"
  DESCRIPTION   "First draft."
 ::= { general 3 }

general OBJECT IDENTIFIER::= { iec62379 1 }

basicUnitStatus OBJECT-IDENTITY
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 6.3.1."
 ::= { generalStatusGroup 1 }

timeOfDayStatus OBJECT-IDENTITY
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 6.3.2."
 ::= { generalStatusGroup 2 }

scheduledOpsStatus OBJECT-IDENTITY
  STATUS        current
  DESCRIPTION   "See IEC 62379-1 section 6.3.3."
 ::= { generalStatusGroup 3 }

END
```

Bibliography

ISO/IEC 8824-2:2002 *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*

ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*

ITU-R Recommendation TF.460:2002, *Standard-frequency and time-signal emissions*

RFC1155, *Structure and identification of management information for TCP/IP-based internets* (IETF Standard #16 STANDARD)

RFC1212, *Concise MIB definitions* (IETF Standard #16 STANDARD)

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

INTERFACE DE COMMANDE COMMUNE POUR PRODUITS AUDIO ET VIDÉO NUMÉRIQUES CONNECTÉS EN RÉSEAUX –

Partie 1: Généralités

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI n'a prévu aucune procédure de marquage valant indication d'approbation et n'engage pas sa responsabilité pour les équipements déclarés conformes à une de ses Publications.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de l'identification de l'un quelconque ou de la totalité de ces droits de propriété industrielle.

La Norme internationale CEI 62379-1 a été établie par le Comité d'étude 100 de la CEI: Systèmes et appareils audio, vidéo et multimédia.

La présente version bilingue (2012-08) correspond à la version anglaise monolingue publiée en 2007-08.

Le texte anglais de cette norme est issu des documents 100/1248/FDIS et 100/1281/RVD.

Le rapport de vote 100/1281/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de maintenance indiquée sur le site web de la CEI sous « http://webstore.iec.ch » dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

0 Introduction

0.1 Vue d'ensemble

Cette famille de normes spécifie un cadre de commande pour le matériel audiovisuel en réseau.

Elle fournit un moyen de gérer des entités permettant de commander non seulement la transmission sur le réseau mais également d'autres fonctions dans des matériels d'interface.

Bien qu'il ait été élaboré à l'origine pour l'audio sur un mode de transfert asynchrone (ATM, en anglais « Asynchronous transfer mode ») en radiodiffusion, le cadre de commande a été étendu de manière à englober la vidéo et d'autres supports à contrainte de temps, ainsi que d'autres technologies de mise en réseau et d'autres applications à la fois dans des environnements professionnels et grand public.

Le cadre de commande fournit un certain nombre de fonctions essentielles:

- il fournit une interface cohérente à la fonctionnalité dans une unité audiovisuelle;
- il permet de bâtir des systèmes réellement « prêts à fonctionner » en donnant aux matériels le moyen de découvrir quelles unités sont reliées au réseau et quelles sont leurs possibilités;
- il associe des zones ou blocs discrets de fonctionnalité d'une façon cohérente et structurée;
- il nous permet de définir des petits blocs de base ciblés à partir desquels une fonctionnalité plus complexe peut être bâtie;
- il assure qu'une nouvelle fonctionnalité peut être mise au point et intégrée de manière cohérente et aisée dans le cadre.

La fonctionnalité fournie par une unité audiovisuelle est représentée en utilisant un ou plusieurs « blocs » (par exemple un commutateur à point de croisement ou une commande de gain) structurés et reliés les uns aux autres à l'aide du cadre de commande.

À titre d'aide complémentaire à la fonctionnalité « prêt à fonctionner », un format commun pour l'audio et la vidéo acheminés sur le réseau est également spécifié, permettant d'éviter les situations dans lesquelles deux éléments matériels ne parviennent pas à communiquer car aucun format n'est pris en charge par les deux. Naturellement, les matériels peuvent également prendre en charge d'autres formats appropriés à des applications particulières et les mécanismes normalisés pour initialiser et terminer une communication fonctionnent pour ces formats de la même manière que pour les formats habituels.

0.2 Structure de la famille de normes

Il est prévu que la CEI 62379 comporte les parties suivantes:

Partie 1: Généralités

Partie 2: Audio

Partie 3: Vidéo

Partie 4: Données

Partie 5: Transmission sur des réseaux

Partie 6: Service de transfert par paquets

La Partie 1 définit les aspects communs à tous les matériels.

Les Parties 2 à 4 définissent la commande des fonctions internes spécifiques à des matériels acheminant des types de supports particuliers; dans le cas de la Partie 4, il s'agit de supports à contrainte de temps autres que l'audio ou la vidéo, par exemple des données RS232 et RS422 pour des applications telles que la commande de machine ou l'état du panneau lumineux « à l'antenne » dans un studio de radiodiffusion. La Partie 4 ne se réfère pas aux données en paquets telles que les messages de commande eux-mêmes.

La Partie 5 définit la commande de transmission de ces supports sur chacune des technologies de réseau, avec une partie secondaire séparée pour chaque technologie. Elle inclut les interfaces de gestion spécifiques ainsi que les éléments de commande spécifiques des réseaux intégrés dans le cadre de commande.

La Partie 6 définit le transport des messages de commande et d'état ainsi que des données non audiovisuelles sur des transports ne prenant pas en charge l'audio et la vidéo, par exemple les liaisons série RS232, avec (comme à la Partie 5) une partie secondaire séparée pour chaque technologie.

0.3 Modèle du matériel commandé

0.3.1 Blocs

On considère qu'un élément matériel (« unité ») est constitué d'éléments fonctionnels ou « blocs » pouvant être reliés entre eux par un routage interne.

Les blocs peuvent posséder des entrées, des sorties et une fonctionnalité interne. L'entrée d'un bloc est généralement connectée à l'entrée du bloc suivant dans la chaîne de traitement. Des paramètres de commande et/ou une surveillance d'état accessible par l'intermédiaire de l'interface de gestion du cadre de commande peuvent être associés aux blocs.

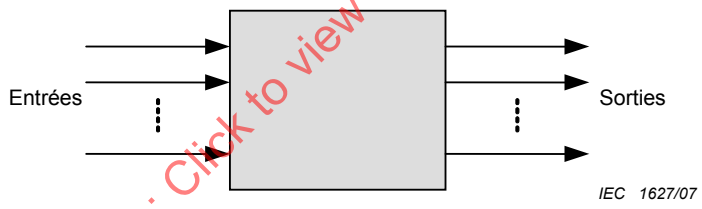


Figure 1 – Bloc

Un bloc type est un préamplificateur ayant une entrée, une sortie et un paramètre de réglage du gain. Un autre bloc est un mélangeur avec plusieurs entrées, une sortie et des paramètres de sélection de la contribution de chaque entrée au mélange; ces paramètres sont effectivement des réglages d'équilibreur. Un générateur de tonalités possède une sortie mais pas d'entrée et des paramètres qui règlent le niveau, la fréquence, etc.

Il existe une catégorie particulière de blocs appelés « accès »; les accès assurent la liaison externe avec les autres matériels. Un « accès d'entrée » est un accès dans lequel de l'audio, de la vidéo ou d'autres données entrent dans l'unité et un « accès de sortie » est un accès d'où ils quittent l'unité. L'accès correspond parfois à une liaison physique, par exemple un connecteur XLR pour audio analogique; c'est parfois une entité virtuelle pouvant être une extrémité d'une connexion sur un réseau ou une voie sur une interface telle que AES10 (MADI) acheminant plusieurs flux audio.

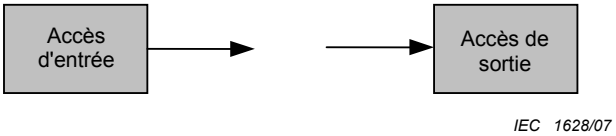


Figure 2 – Accès

Un accès d'entrée n'a pas d'entrée (ou plutôt pas d'entrée interne; il possède une entrée externe, mais ne faisant pas partie du modèle de la structure interne de l'unité) et une sortie unique, fournissant le flux entrant aux entrées des autres blocs. Dans le cas d'un accès au réseau, des paramètres définissent l'adresse du réseau; un accès audio physique peut comporter des paramètres indiquant la vitesse d'échantillonnage et la profondeur en bits. De façon similaire, un accès de sortie possède une seule entrée et aucune sortie (interne).

La Figure 3 montre un exemple de la façon dont les divers blocs sont reliés dans une unité. Noter que chaque entrée est reliée exactement à une sortie mais qu'une sortie peut être reliée à plusieurs entrées ou à aucune.

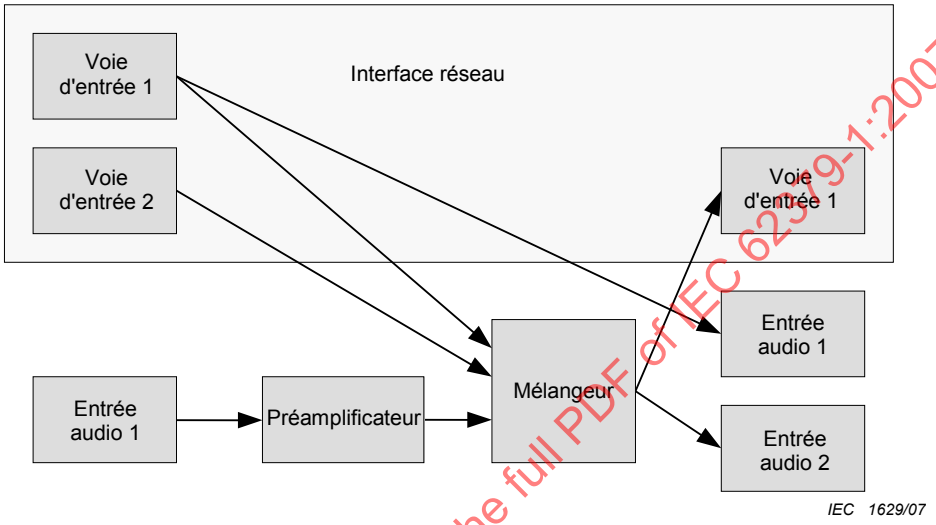


Figure 3 – Exemple d'« unité »

Il existe un bloc réalisant un mélange entre trois entrées, deux du réseau et une d'un accès audio physique (ou, vu sous un autre angle, deux de sources distantes et une d'une source locale). La source locale est raccordée par l'intermédiaire d'un préamplificateur. Le signal résultant est fourni localement en sortie sur la sortie 2 et est également transmis sur le réseau. Une autre sortie locale achemine une copie de l'une des sources distantes.

L'ensemble de blocs disponibles, la connectivité entre eux et les paramétrages de chacun peuvent être fixes ou modifiables au moyen d'un terminal de gestion ou en lecture seule mais modifiables par des facteurs externes. Lorsque des blocs sont mis en œuvre par logiciel, une unité peut permettre à un terminal de gestion de les créer et de les effacer. Lorsque des blocs sont mis en œuvre par des circuits physiques, les blocs eux-mêmes ne peuvent pas être modifiés mais il est toujours possible au terminal de gestion de reprogrammer le routage entre eux.

0.3.2 Cadre de commande

Le cadre de commande est constitué de deux listes; une liste de blocs (appelés également éléments de commande) et une liste de liaisons entre eux. Dans les deux listes, un bloc individuel est identifié par un « identifiant de bloc » qui est un nombre différent pour chaque bloc d'une unité.

L'entrée d'un bloc dans la liste de blocs montre de quel type de bloc il s'agit, représenté par une valeur globalement unique, comme décrit en 0.3.5.

Les groupes de blocs reliés entre eux sont appelés chaînes de traitement. Une chaîne de traitement représente généralement ce qu'effectue une unité dans son ensemble, ainsi par exemple une unité modifiant le gain d'une entrée pour produire une sortie comporte une seule

chaîne de traitement constituée d'un accès d'entrée relié à un bloc de gain, lui-même relié à un accès de sortie.

0.3.3 Façon dont le cadre facilite la conception des unités

La présente norme prévoit qu'un grand nombre de blocs de commande vont être conçus et mis en œuvre dans l'avenir pour commander le grand nombre de types différents de fonctionnalités procurées par les unités audio et audiovisuelles.

Des unités peuvent être réalisées en partant de blocs existants ou de nouvelles peuvent être créées selon les besoins. Il est souvent possible de représenter des fonctionnalités de commande complexes spécifiques à un produit en utilisant un certain nombre d'instances liées de blocs de base plus simples fournissant ensemble la fonctionnalité requise.

0.3.4 Façon dont le cadre autorise le « prêt à fonctionner »

Un terminal de gestion a simplement besoin de reconnaître les blocs qui correspondent aux fonctions qu'il commande. (Le terme « terminal de gestion » recouvre une large diversité de matériels, d'un système de commande de diffusion jusqu'à l'interface utilisateur d'un dispositif sur un réseau domestique.)

Il peut découvrir les unités qui sont présentes sur le réseau et les fonctions que chacune contient; il n'a pas besoin de reconnaître les unités elles-mêmes mais uniquement les blocs qui décrivent la fonctionnalité qui l'intéresse.

Le processus de découverte consiste à:

- créer une liste des unités, en commençant par celle auxquelles il est directement connecté; les unités peuvent être identifiées de manière unique par leur adresse MAC sur 48 bits;
- récupérer la liste de blocs de chaque dispositif de la liste; s'il en existe qui sont des accès réseau donnant accès à d'autres dispositifs, ajouter ces dispositifs à la liste (sauf s'ils se trouvent déjà sur celle-ci);
- récupérer la connectivité entre tous les blocs pour lesquels elle est appropriée.

Par exemple, l'interface utilisateur avec un système audio à son d'ambiance peut rechercher des unités contenant des sources audio, trouver celles pour lesquelles existe une chaîne de traitement leur permettant de les rendre disponibles à l'utilisateur et les proposer dans un menu. Elle identifie également des fonctions de la chaîne de traitement telles que la commande de volume et les commandes de lecture (pause, retour arrière, saut de piste, etc.).

Dans un système de commande de diffusion, un processus similaire peut être exécuté lorsque le système est installé et à tout moment lorsque du matériel est ajouté ou remplacé. Ce processus est sous le contrôle de l'installateur, plutôt que de s'exécuter automatiquement, mais il convient au moins qu'il soulage l'installateur de la nécessité de saisir des adresses réseau.

0.3.5 Définition d'un nouveau type de bloc

Une entrée de bloc dans la liste de blocs montre le type de bloc dont il s'agit, représenté par un identifiant d'objet (OID) (voir 0.4.2) qui est une valeur globalement unique identifiant la définition du type de bloc.

L'exigence principale lors de l'ajout d'un nouveau type de bloc au cadre de commande est que sa définition de type de bloc soit conforme aux conditions suivantes:

- l'OID globalement unique identifie un tableau de MIB (Base d'informations de gestion, en anglais « Management information base ») ou un groupe de tableaux de MIB, chaque tableau contenant un nombre variable de rangées.

- le ou les tableaux sont indexés en utilisant l'identifiant de bloc pour accéder à des objets de commande associés à des instances individuelles de ce type de bloc.

En effet, le cadre fournit le point d'entrée pour commander chaque bloc de fonctionnalité. Il sera toujours nécessaire de spécifier individuellement les détails réels concernant la façon de commander cette fonctionnalité.

0.4 Base d'informations de gestion (MIB, en anglais « Management information base »)

0.4.1 Objets

La communication entre le terminal de gestion et l'unité gérée s'effectue par le protocole de gestion simple de réseau (SNMP), qui définit toutes les opérations de gestion en termes de lectures et d'écritures sur une collection de variables hiérarchiquement organisées ou « objets gérés », appelée MIB (Base d'informations de gestion, en anglais « Management information base »).

Chaque objet est identifié par un OID (Identifiant d'objet, en anglais « Object identifier »), qui est une séquence de nombres; dans la description, ils sont séparés par des points et il existe également des noms alphanumériques pouvant être utilisés en remplacement. Les identifiant des objets définis dans une norme de cette famille commencent par 1.0.62379.*p* s'ils sont définis dans la partie *p*, ou 1.0.62379.*p.s* s'ils sont définis dans la partie *p-s*.

Chaque bloc est décrit par un groupe d'objets (« enregistrement »). Ces objets sont les paramètres de commande et les variables d'état. Pour chaque type de bloc, la structure de l'enregistrement est spécifiée dans l'une des parties de la CEI 62379 ou ailleurs (pour les blocs spécifiques à un produit ou spécifiques à une application). La connectivité est décrite par un tableau contenant une identification de la sortie à laquelle chaque entrée est raccordée (voir `connectorTable` en 4.2.5).

Ainsi, le terminal de gestion peut découvrir la fonctionnalité que possède une unité et il peut être capable d'en reprogrammer une partie. Le protocole SNMP impose qu'une commande de modification d'un objet soit toujours confirmée par un message présentant la nouvelle valeur de l'objet, de sorte que si une unité ne prend pas en charge toute la gamme des valeurs possibles d'un paramètre, il peut choisir la valeur la plus proche de la valeur demandée et indiquer en retour au terminal de gestion exactement ce qui a été effectué.

0.4.2 Autres utilisations des OID (Identifiants d'objet, en anglais « Object identifiers »)

Les OID (Identifiants d'objet, en anglais « Object identifiers ») sont parfois utilisés pour identifier des éléments autres que des objets. Chaque type de bloc est représenté par un OID; dans ce cas, il constitue également la racine (les quelques premiers nombres de la séquence) des OID pour tous les objets de l'enregistrement qui décrivent chaque bloc de ce type.

Les OID ne sont pas limités à ceux qui sont spécifiés dans les diverses parties de la CEI 62379; pour les blocs spécifiques à des produits, les responsables de la mise en œuvre peuvent définir d'autres types, par exemple, avec des OID de racine 1.3.6.1.4.1.*n*, où *n* est le numéro d'entreprise du fabricant de l'unité. Un terminal de gestion à usage général, reconnaissant uniquement les OID normalisés considère ces blocs spécifiques à un produit comme des « boîtes noires »; il reconnaît leur connectivité mais n'est pas capable de commander ou de surveiller leur fonctionnement.

Les formats des supports (audio, vidéo, etc.) sont également identifiés par des OID. Ici encore, des OID dont la racine ne se trouve pas en 1.0.62379 peuvent être utilisés pour des formats qui ne sont pas définis dans cette famille de normes; à condition qu'à la fois les dispositifs d'émission et de réception prennent en charge le format, un appel peut être lancé

sans que le terminal de gestion soit capable de le reconnaître, le terminal de gestion peut donc ne pas être capable d'afficher une description conviviale de celui-ci.

0.4.3 Migration vers XML

XML est de plus en plus utilisé comme interface avec des applications de gestion de réseau. Toutefois, la communication avec les agents de gestion dans les dispositifs en réseau s'effectue habituellement par l'intermédiaire d'une passerelle effectuant une traduction entre SNMP et XML, de sorte que seule l'unité gérée doit prendre en charge SNMP.

Une addition future à la CEI 62379 (amendement ou partie supplémentaire) pourra normaliser la forme de XML; toutefois, dans ce cas, les objets gérés sous-jacents resteront les mêmes dans les deux formes.

0.5 Diffusions d'état

0.5.1 Introduction

Les pages d'état sont des messages contenant des valeurs structurées représentant un état interne d'une unité. Chaque page est organisé dans un format fixe d'informations associées. Une unité peut définir et prendre en charge plusieurs types de page d'état.

Les pages d'états associées sont rassemblées en groupes appelés groupes de diffusions d'état. Lorsqu'une entité distante nécessite une diffusion d'état, elle spécifie le groupe qu'elle souhaite recevoir. Les groupes de diffusion d'état sont identifiés par des OID (Identifiants d'objet, en anglais « Object identifiers »).

Les pages d'état constituent la méthode préférentielle de surveillance régulière de l'état d'une unité. Les champs de scrutation dans la MIB (Base d'informations de gestion, en anglais « Management information base ») imposent une charge supplémentaire à la fois à l'unité et au réseau, car ils nécessitent que l'unité traite une demande SNMP à chaque occasion, plutôt que d'effectuer la diffusion une seule fois. Si les mêmes informations sont requises à des emplacements multiples, il existe une demande séparée de chacun et plusieurs copies des informations doivent être envoyées, tandis qu'avec la diffusion, une seule copie est envoyée, dupliquée par le réseau selon les besoins pour atteindre toutes ses destinations.

La présente norme définit un certain nombre de pages d'état et de groupes. D'autres parties définissent leurs propres pages d'état et groupes. Les fabricants peuvent également définir des pages d'état et des groupes spécifiques à un produit.

0.5.2 Sources d'informations de page d'état

Les pages d'état et les groupes peuvent être associés soit à un bloc unique, soit à l'unité dans son ensemble.

Trois informations sont nécessaires pour initialiser un appel de diffusion de page d'état (toutefois, d'autres informations peuvent être nécessaires, celle-ci sont spécifiques au réseau et sont spécifiées dans la partie secondaire correspondante de la Partie 5):

- identifiant de bloc, du bloc devant être la source de l'appel de groupe de diffusion d'état (une valeur nulle est utilisée pour les diffusions d'état associées à l'unité dans son ensemble);
- OID (Identifiant d'objet, en anglais « Object identifier ») de groupe de pages, du groupe de diffusion d'état particulier à produire;
- vitesse de page requise, vitesse à laquelle les pages sont générées.

Si une unité prend en charge plusieurs instances d'un type de bloc (par exemple, une sortie audio AES3 prenant en charge un groupe de diffusions d'état de niveau audio), il n'est pas

nécessaire de mettre en œuvre le groupe de diffusions d'état associé pour chaque instance, il peut être mis en œuvre simplement pour un de leurs sous-ensembles.

0.5.3 Format général d'une page d'état

Les deux premiers octets d'une page d'état indiquent le type de page. Il est nécessaire que les identifiants de type de page soient uniques pour toutes les pages faisant partie d'un groupe de pages donné, mais ils peuvent dans le cas contraire être librement alloués par l'organisme ou le fabricant spécifiant le contenu des pages. Le format du reste de la page dépend du type de page; la longueur maximale est de 484 octets.

Les nombres sont codés en binaire en utilisant un nombre d'octets fixe et big-endian. Les chaînes textuelles sont en UTF-8.

0.6 Appels

Il est prévu que le réseau fournisse les deux types de services suivants.

- Débit binaire fixe d'un service vers un grand nombre pour les flux de support et les diffusions d'état, avec une latence et un débit garantis.
- « Service au mieux » de transfert biunivoque bidirectionnel de paquets pour les messages de commande.

Sur un réseau ATM (Mode de transfert asynchrone, en anglais « Asynchronous transfer mode »), ceux-ci sont mappés respectivement sur les appels point à multipoint CBR et point à point UBR.

Sur un réseau sans connexion (et sans état) tel que IP, il n'existe pas d'équivalent direct au concept « d'appel » ATM. Toutefois, si le réseau doit fournir les garanties nécessaires de qualité de service (QoS) pour les flux de support, il est nécessaire qu'il existe un certain type de mécanisme pour allouer les ressources nécessaires pour un flux. Dans un grand nombre de cas, ce mécanisme doit être similaire au processus d'établissement d'appel des réseaux orientés connexion.

Dans le cas du service de transfert de paquets, qui peut parfaitement être sans connexion au niveau de la couche réseau, il existe pour un grand nombre d'applications une relation entre le terminal de gestion et l'unité gérée à l'intérieur de laquelle certaines informations « d'état » persistent entre les transactions. Par exemple, lors de la mise à jour d'un logiciel dans l'unité (voir 0.9), un seul terminal de gestion peut écrire dans une zone mémoire donnée à la fois. Ainsi, il existe une « session » entre le terminal de gestion et l'unité gérée, correspondant à un appel sur un réseau orienté connexion.

À tout flux de support, sont associées des informations « d'identité d'appel » pouvant inclure des informations concernant le flux lui-même, le point où il entre dans le système et chaque destination. Une « importance » est assignée aux destinations et il s'agit normalement seulement des destinations les plus importantes rapportées. De plus amples détails sont donnés à l'Article A.4.

0.7 Niveaux de privilège

Dans l'ensemble de la CEI 62379, le concept de « niveaux de privilège » est utilisé pour distinguer différents types d'utilisateurs. Ce concept a été élaboré pour les studios de radiodiffusion, et les noms des niveaux le représentent mais sont également utiles dans d'autres applications.

Quatre niveaux de privilège sont définis:

- auditeur;
- opérateur;

- superviseur;
- maintenance.

L'auditeur est le niveau de privilège le plus bas, destiné à être utilisé par les personnes qui souhaitent surveiller les signaux audio ou vidéo traversant l'unité (par exemple, quelqu'un souhaitant écouter la sortie d'un studio par l'intermédiaire de son ordinateur personnel). Les auditeurs peuvent établir des appels à partir de sources audio et vidéo vers le matériel local pour ceux-ci mais ils ne peuvent rien modifier pouvant avoir une influence sur ce que reçoivent les autres utilisateurs.

L'opérateur est le niveau de privilège suivant, destiné aux personnes qui commandent le fonctionnement quotidien de l'unité (par exemple, un technicien de studio). Les opérateurs peuvent modifier des éléments ayant un impact sur les autres utilisateurs, par exemple le mélange de signaux fournissant la sortie d'un studio ou la fourniture de commandes « pause », « recherche », etc. au matériel de lecture.

Le superviseur est le niveau de privilège suivant, destiné à être utilisé par les personnes qui commandent et gèrent le réseau, par exemple un technicien de salle de contrôle.

La maintenance est le niveau de privilège le plus haut, destiné aux personnes qui ont besoin d'exécuter des tâches pouvant interrompre le fonctionnement normal de l'unité, par exemple une mise à jour de progiciel ou l'entrée de l'unité dans un mode de diagnostic.

Les niveaux de privilège sont destinés à être utilisés pour deux objectifs. Le premier consiste à réserver la capacité d'appel de gestion à chaque catégorie de demandeur, pour empêcher le blocage d'appels de gestion importants du fait qu'un trop grand nombre d'appels de niveau inférieur sont en cours. Le deuxième consiste à empêcher les demandeurs d'exécuter des tâches de commande inappropriées en limitant les commandes acceptées aux niveaux inférieurs.

Pour autoriser cette dernière fonctionnalité, à chaque appel est associé un niveau de privilège et un appel ne peut pas être établi, modifié ou démonté par un appel de gestion de privilège inférieur. Un opérateur peut par exemple établir des appels avec le privilège d'opérateur, appels qui ne peuvent plus ensuite être modifiés par des appels de gestion ayant uniquement un privilège d'auditeur, bien qu'ils puissent être modifiés ou interrompus par d'autres opérateurs et par des superviseurs. Les superviseurs peuvent établir des appels que ni les auditeurs ni les opérateurs ne peuvent perturber; la liaison entre un studio de continuité et la chaîne de transmission peut en constituer un exemple.

La spécification nécessite qu'au moins un appel de gestion puisse être accepté à chaque niveau de privilège à un instant donné quelconque. Dans la pratique, la capacité d'appel qu'il est nécessaire de réserver pour chaque niveau dépend vraisemblablement du contexte à l'intérieur duquel l'unité est utilisée, de sorte qu'un moyen de le configurer est également spécifié. Il est nécessaire d'allouer toute capacité d'appel non réservé sous la forme premier arrivé, premier servi.

0.8 Automatisation

Le mécanisme d'automatisation permet de déterminer une seule ou plusieurs valeurs à un instant donné. L'aménagement réel utilise une liste d'événements d'automatisation où chaque événement est constitué de l'heure, de l'OID (Identifiant d'objet, en anglais « Object identifier ») d'un objet dans la MIB (Base d'informations de gestion, en anglais « Management information base ») et de la valeur à entrer à ce moment.

Ceci élimine l'incertitude introduite par le service au mieux sur le réseau, le contrôleur peut ajouter l'événement au tableau suffisamment longtemps à l'avance pour laisser le temps à la demande d'être répétée si celle-ci est perdue. Il peut également programmer l'apparition simultanée d'un certain nombre d'événements.

Par ailleurs, des événements multiples peuvent être associés de sorte qu'ils se produisent les uns après les autres, ce qui ressemble davantage à une macro-commande.

0.9 Téléchargement de logiciel

En 4.2.8 et 5.2, la présente norme comporte un mécanisme de mise à jour de logiciel et d'autres informations de configuration dans des unités prenant en charge l'interface de commande commune.

Le but est qu'un terminal de gestion soit capable de mettre à jour un logiciel dans un grand nombre de matériels différents provenant de fabricants différents sans qu'il soit nécessaire d'exécuter des applications spécifiques à un fabricant.

Les objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») définis en 4.2.8 sont destinés à constituer un modèle commun à l'ensemble des divers types de mémoires pouvant être utilisées dans l'unité gérée. Une unité peut contenir plusieurs « classes » de mémoire; des classes différentes peuvent être physiquement différentes, par exemple, une mémoire flash et une mémoire magnétique rotative et/ou réservées pour différents types de données, par exemple, un logiciel et des fichiers audio.

EXEMPLE 1: Système simple utilisant une mémoire flash

Une mémoire flash est constituée de blocs ayant généralement chacun 64 ko. Un octet individuel peut être écrit à condition que ceci n'entraîne pas une modification d'un quelconque bit de 0 à 1. Un bloc complet peut être « effacé », après quoi, chaque bit du bloc est à 1.

Chaque zone est constituée soit d'un bloc unique soit de plusieurs blocs adjacents; quelques octets sont réservés pour contenir la longueur, le type et le numéro de série. Le « pointeur » qui identifie une zone est la partie haute de l'adresse du premier octet de la zone.

La suppression s'effectue par effacement de tous les blocs constituant la zone, ce qui peut prendre plusieurs secondes pour certaines puces de mémoire flash. Après effacement, s'il existe une zone vide adjacente, les deux zones sont fusionnées de manière à former une zone vide unique (de sorte que l'une d'entre elle disparaît du tableau).

S'il n'y a pas d'autre mémoire dans laquelle des composants peuvent être chargés, il convient que toutes les zones soient de classe 1.

EXEMPLE 2: Disque avec système de remplissage

Chaque fichier est une « zone » et il existe une autre zone (unique) contenant la totalité de l'espace libre. Dans le cas d'un système de remplissage Unix, le « pointeur » peut être le numéro inode du fichier.

Si le produit contient à la fois un disque et une mémoire flash, il peut identifier la mémoire flash comme classe 2 et le disque comme classe 1 ou il peut diviser le disque en partitions séparées identifiées comme classes 3, 4, etc.

Un objet pour chaque zone représente l'`access` autorisé, c'est-à-dire s'il peut être écrit et/ou effacé. Le fait qu'une zone soit incluse dans le tableau et l'accès qui lui est autorisé peuvent dépendre du niveau de privilège. Le terminal de gestion peut être capable de modifier l'`access`, mais ceci est habituellement commandé par l'unité gérée; elle peut par exemple régler toutes les zones nécessaires pour charger le logiciel en cours d'exécution en lecture seule (c'est-à-dire, ne pouvant pas être écrites) et le reste en accès complet, de sorte qu'un nouveau logiciel peut être téléchargé dans les zones actuellement inutilisées, mais le logiciel en cours ne peut pas être écrasé avant que le nouveau logiciel ait été entièrement chargé.

Un autre est le `status` qui indique l'opération qui est en cours d'exécution sur la zone. Le terminal de gestion demande la suppression d'une zone en fixant son `status` à « effacement », l'unité gérée supprime alors son contenu courant et règle le `status` à « vide ». Il peut ensuite l'amalgamer avec une autre zone vide dans la même classe de mémoire.

Une zone possède également un `serialNumber`; un nouveau logiciel reçoit le numéro de série suivant en séquence après celui du logiciel en cours. Lorsque l'unité est redémarrée, le logiciel (valable) dont le numéro de série est le plus récent est chargé; les procédures de 5.2.3 assurent qu'un logiciel partiellement chargé n'est pas valable, par exemple, si l'unité est redémarrée avant que le processus de téléchargement soit terminé. Ainsi, si l'unité est redémarrée avant que le nouveau logiciel n'ait été entièrement chargé, l'ancien logiciel est exécuté; si elle est redémarrée après, le nouveau logiciel est exécuté.

Deux méthodes de distribution de logiciel sont prises en charge. Le logiciel peut être fourni sous forme d'un paquet, par exemple sur un CD ou par envoi d'un fichier ZIP par courrier électronique, comme spécifié en 5.2.6.1; lorsqu'un nouveau logiciel est disponible, un nouveau paquet doit être distribué. En variante, un « fichier de produit » contenant un URL peut être fourni et le logiciel téléchargé sur Internet, comme spécifié en 5.2.6.2. Ceci facilite la vérification de la disponibilité du nouveau logiciel par le terminal de gestion.

0.10 Encapsulation de messages

Dans l'ensemble de cette famille de normes, le terme « message » est utilisé pour signifier une chaîne d'octets qui est acheminée sur le réseau sous forme d'une unité simple. Sur un réseau IP, il s'agit de la partie « données » d'un paquet ou datagramme, c'est-à-dire à l'exclusion des en-têtes et de la mise en trame. Dans le cas d'un réseau ATM (Mode de transfert asynchrone, en anglais « Asynchronous transfer mode »), il s'agit normalement d'un AAL5-SDU.

Ainsi, sur un réseau IP, les messages SNMP sont mis en paquets de la manière normale pour le SNMP, c'est-à-dire en utilisant UDP sur IP. Sur un réseau ATM ils sont mis directement en paquets en AAL5, de la même manière que dans ILMI.

0.11 Autres informations

Des explications supplémentaires de certains aspects de la présente norme sont données à l'Annexe A.

INTERFACE DE COMMANDE COMMUNE POUR PRODUITS AUDIO ET VIDÉO NUMÉRIQUES CONNECTÉS EN RÉSEAUX –

Partie 1: Généralités

1 Domaine d'application

La présente partie de la CEI 62379 spécifie une interface de commande pour des produits transmettant des signaux audio et/ou vidéo sur des réseaux numériques.

Des documents séparés précisent les éléments spécifiques à un type particulier de trafic, une technologie particulière de mise en réseau ou une classe particulière d'application.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

ISO/CEI 646:1991, *Technologies de l'information – Jeu ISO de caractères codés à 7 éléments pour l'échange d'informations*

ISO/CEI 8824-1:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): Spécification de la notation de base*

IEEE Std 802:2001, *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*

RFC1157, *Simple Network Management Protocol (SNMP)* (IETF Standard #15 STANDARD)

RFC 1441, *Introduction to version 2 of the Internet-standard Network Management Framework (IETF)*

RFC 3411-3418, *Simple Network Management Protocol version 3* (IETF Standard #62)

3 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.1
temps universel coordonné
UTC
échelle de temps constituant la base d'une dissémination coordonnée de fréquences et de signaux horaires étalons (voir la Recommandation UIT-R TF.460)

3.2
appel
appel point à point ou appel point à multipoint

3.3
remplacement d'appel
processus au moyen duquel un nouvel appel est établi pour remplacer un appel existant, le nouvel appel suivant une route différente dans le réseau et les changements de destination des cellules réceptrices sur l'ancien appel pour les recevoir sur le nouvel appel sans interruption du flux de données transporté par celles-ci

3.4
système mondial de radiorepérage
GPS
source de données horaires disponible dans le monde entier d'après laquelle on peut déterminer l'UTC; également aide à la navigation

3.5
adresse MAC
adresse MAC de LAN de 48 bits, comme spécifié en 9.2 de l'IEEE 802, identifiant un point de liaison d'une unité à un réseau

3.6
message
chaîne d'octets acheminée sur le réseau sous forme d'une simple unité

3.7
horloge en temps réel non volatile
composant fournissant l'heure du jour et la date et continuant ainsi en l'absence de référence externe, même si l'unité qui le contient n'a pas été alimentée en continu depuis la dernière fois où une référence externe était disponible

3.8
octet
groupe de 8 bits

3.9
identificateur unique d'organisation
OUI
valeur globalement unique de 24 bits assignée à une entreprise ou à un autre organisme, destinée à être utilisée dans des adresses MAC et d'autres identifiants, voire l'Article 9 de l'IEEE 802

3.10
appel point à multipoint
trajet unidirectionnel sur un réseau acheminant des informations d'une unité de source unique à une ou plusieurs unités de destination, les informations étant copiées par le réseau à chaque fois que le trajet se subdivise

3.11
appel point à point
trajet bidirectionnel sur un réseau acheminant des informations dans les deux directions entre deux unités d'interface du réseau

3.12
accès
entrée extérieure vers ou sortie depuis une unité

3.13
unité
élément simple d'un matériel

3.14 Autres abréviations

3.14.1
Audio Engineering Society
AES

3.14.2
notation de syntaxe abstraite numéro un
ASN.1

3.14.3
règles de codage de base
BER

3.14.4
radiodiffusion numérique
DAB

3.14.5
Internet Assigned Numbers Authority
IANA

3.14.6
groupe d'étude sur l'ingénierie Internet
IETF

3.14.7
Protocole Internet
IP

3.14.8
Union Internationale des Télécommunications
UIT

3.14.9
secteur de la normalisation des télécommunications de l'UIT
ITU-T

3.14.10
réseau local
LAN

3.14.11
contrôle d'accès au support
MAC

3.14.12
base d'informations de gestion
MIB

3.14.13
Groupes d'experts en images animées
MPEG

3.14.14
système de radiodiffusion de données
RDS

3.14.15
protocole de gestion de réseau simple
SNMP

3.14.16
Localisateur universel de ressources
URL

3.14.17
Forme de codage du format de transformation unicode 8 bits
UTF-8

4 Gestion d'unité

4.1 Protocole

Le protocole utilisé pour la gestion d'unité doit être SNMP, soit la version 1, telle que spécifiée dans l'IETF RFC1157 soit la version 2, telle que spécifié dans l'IETF RFC1441, soit la version 3, telle que spécifié dans l'IETF RFC3411-3418.

Si l'on utilise la version 1 de SNMP, les références dans cette norme au message RESPONSE doivent être constituées de référence au message GET RESPONSE.

Chaque message doit être encapsulé sous forme d'une unité de données de service pour le service de transfert de paquets fourni par le réseau sur lequel il est transmis.

NOTE 1 Les détails du service de transfert de paquets sont spécifiés dans la partie secondaire de la CEI 62379-5 ou de la CEI 62379-6 s'appliquant au réseau ou à la liaison qui le fournit.

NOTE 2 La RFC1157 spécifie qu'il faut que toutes les unités prennent en charge des messages d'une longueur allant jusqu'à 484 octets; si le service de transfert de paquets comporte la négociation de la longueur maximale des messages, des messages plus longs peuvent être pris en charge.

Une unité gérée doit prendre en charge au moins une version de SNMP; un terminal de gestion doit prendre charge toutes les versions.

NOTE 3 Le terminal de gestion peut déterminer la version prise en charge par l'unité gérée par signalisation pendant l'établissement de l'appel ou simplement par approximations successives.

NOTE 4 Si une authentification est exigée, il convient d'utiliser la version 3 (plutôt que d'utiliser les noms de communauté de la v1 ou de la v2). Il en est de même si un chiffrement est exigé.

4.2 Définitions de la MIB (Base d'informations de gestion, en anglais « Management information base »)

4.2.1 Généralités

Toutes les fonctions de commande gérées par le réseau dans un matériel conforme doivent être représentées par des instances de types d'objets gérés définies dans la présente norme ou dans d'autres parties de cette norme ou ailleurs.

NOTE 1 L'utilisation des types d'objets gérés qui ne sont pas définis par la CEI 62379 est autorisée pour permettre la commande de fonctions ne faisant pas partie du domaine d'application de la CEI 62379 utilisant le protocole normalisé. Le cas échéant, il convient d'utiliser les types d'objet définis par la CEI 62379 de préférence aux types d'objet définis ailleurs.

NOTE 2 Les objets gérés sont généralement organisés en groupes de fonctions associées.

Chaque type d'objet géré dans cette partie de la CEI 62379 ou dans les autres est défini par les attributs suivants.

- *Identifier* spécifie le nom et le numéro identifiant l'objet par rapport au groupe ou l'objet dont il descend.
- *Syntax* spécifie la syntaxe de la structure de données abstraite représentant la valeur de l'objet. La syntaxe est décrite en utilisant la notation de syntaxe abstraite 1 (ASN.1), comme spécifié dans l'ISO/CEI 8824-1:1998.
- *Index* spécifie si l'objet est utilisé pour identifier de manière unique une rangée dans un tableau d'objets gérés.
- *Readable* spécifie le niveau de privilège (tel que défini ci-dessous) pour un accès en lecture à l'objet. Un appel de gestion avec un niveau de privilège supérieur ou égal à ce niveau doit être autorisé pour effectuer une opération get ou get-next sur l'objet. Un niveau d'accès en lecture de *none* spécifie que les opérations get et get-next ne sont pas autorisées sur l'objet.
- *Writable* spécifie le niveau de privilège (tel que défini ci-dessous) pour un accès en écriture à l'objet. Un appel de gestion avec un niveau de privilège supérieur ou égal à ce niveau doit être autorisé pour effectuer une opération set sur l'objet. Un niveau d'accès en écriture de *none* spécifie que les opérations set ne sont pas autorisées sur l'objet.
- *Volatile* spécifie si la valeur courante de l'objet est conservée après une réinitialisation matérielle ou une période de perte d'alimentation:
  - *no* indique que la valeur de l'objet est incorporée dans l'unité ou enregistrée dans une mémoire non volatile;
  - *yes* indique que la valeur de l'objet est transitoire;
  - *maybe* indique que la valeur de l'objet peut ou non être volatile, en fonction de la conception de l'unité ou de la valeur d'un certain objet dans la MIB.
- *Status* spécifie le niveau de prise en charge de mise en œuvre requis pour l'objet:
  - *mandatory* indique que l'objet doit être mis en œuvre par tout le matériel conforme mettant également en œuvre le groupe parent de cet objet ou l'objet;
  - *optional* indique que l'objet peut ou non être mis en œuvre par tout matériel conforme mettant également en œuvre le groupe parent de cet objet ou l'objet.
  - *deprecated* indique que l'objet ne doit être mis en œuvre par aucun matériel conforme nouvellement conçu.

Pour simplifier, les attributs d'états sont abrégés par *m*, *o*, et *d* dans les tableaux de définition d'objets.

Les niveaux de privilège utilisés pour les attributs *readable* et *writable* sont du plus bas au plus haut:

- *listener*

- *operator*
- *supervisor*
- *maintenance*
- *none*

NOTE 3 Les types d'objets gérés sont définis dans la présente norme sous une forme relativement concise, lisible par un être humain. Chaque partie de la présente norme fournit également une version lisible par une machine des définitions qui y sont contenues, dans une annexe informative. Les définitions lisibles par une machine sont décrites en utilisant la structure pour les informations de gestion version 2 (SMIv2), spécifiée dans l'IETF STD 58. Noter que SMIv2 peut également représenter un sous-ensemble des attributs ci-dessus.

4.2.2 Définitions de type au niveau de l'application

Les types suivants au niveau de l'application sont utilisés pour spécifier la syntaxe des objets gérés à la fois dans la présente norme et dans d'autres parties de celle-ci.

NOTE 1 Les trois types suivants sont utilisés pour garantir la compatibilité avec SMIv2, nécessitant que toutes les valeurs entières puissent être représentées sur 32 bits et imposant des restrictions supplémentaires aux valeurs entières utilisées pour indexer les tableaux. Ils n'ont pas pour but d'imposer au matériel d'accepter ou de mémoriser toute la gamme de valeurs.

```
IntegerNumber::= INTEGER (-2147483648..2147483647)
-- Valeur entière sans restriction.

CardinalNumber::= INTEGER (0..2147483647 )
-- valeur entière positive ou nulle.

IndexNumber::= INTEGER (1..2147483647)
-- valeur entière positive.
```

NOTE 2 Un `IndexNumber` est ainsi un entier pouvant être utilisé pour indexer un tableau. Ce type est utilisé lorsque la numérotation est consécutive ascendante à partir de 1. Dans les autres cas, un type de « pointeur », tel que `ClockSource` ou `SoftwareArea`, est défini.

NOTE 3 Les deux types suivants sont également utilisés pour garantir la compatibilité avec SMIv2, ne permettant pas d'utiliser les types ASN.1 normalisés `BOOLEAN` et `UTF8String`. Le type `TruthValue` mappe directement sur le type du même nom défini dans SMIv2.

```
TruthValue::= INTEGER {
  true  (1),
  false (2)
} (true..false)
-- Énumération représentant une valeur booléenne.

Utf8String::= OCTET STRING (SIZE(0..80))
-- Chaîne textuelle codée UTF-8.

Octet::= OCTET STRING (SIZE(1))
-- Valeur arbitraire pouvant être représentée sur 8 bits.

BlockId::= INTEGER (1..2147483647)
--  Pointeur identifiant de manière unique un bloc de commande dans
            l'unité.

BlockType::= OBJECT IDENTIFIER
-- Identifiant d'objet identifiant un bloc de commande défini. Le bloc
-- peut être un bloc défini dans une quelconque partie de la CEI 62379 ou
            un bloc défini ailleurs.

MediaFormat::= OBJECT IDENTIFIER
-- Identifiant d'objet identifiant un format de support défini. Le format
-- peut être un bloc défini dans une quelconque partie de la CEI 62379 ou
            un bloc défini ailleurs.
```

```
PortDirection::= INTEGER {
  input (1),
  output (2)
} (input..output)
-- Énumération identifiant si un accès est une entrée ou une sortie.

StatusSource::= INTEGER {
  unitWide (0)
} (unitWide | BlockId)
-- Pointeur identifiant de manière unique une source de diffusion d'état
              dans l'unité.

SourceBlockId::= INTEGER {
  nullBlock  (0)
} (nullBlock | BlockId)
--  Pointeur identifiant de manière unique un bloc de commande dans
              l'unité qui
-- peut également prendre une valeur nulle.

PrivilegeLevel::= INTEGER {
  listener    (1),
  operator    (2),
  supervisor  (3),
  maintenance (4)
} (listener..maintenance)
-- Énumération identifiant un niveau de privilège d'appel.
```

Les types suivants au niveau de l'application sont utilisés pour spécifier la syntaxe des objets gérés définis dans la présente norme.

```
MacAddress::= OCTET STRING (SIZE(6))
-- Adresse de 48 bits à partir de l'espace de nombre utilisé pour les
              adresses MAC IEEE802.
-- Voir 9.2 de l'IEEE Std 802-2001

TDomain::= OBJECT IDENTIFIER
-- Identifiant d'objet identifiant un type d'adresse de réseau défini. Le
-- type d'adresse peut être un type défini dans une quelconque partie de
              la CEI 62379 ou un
-- type défini ailleurs.

TAddress::= OCTET STRING (SIZE(1..255))
-- Adresse spécifique au réseau. Il convient de définir explicitement la
              sémantique
-- par un objet de la MIB associé mais elle peut également être définie
              implicitement par
-- les capacités du matériel.

UnitIdentity::= OCTET STRING (SIZE(10))
-- Valeur de code identifiant de manière unique le type de matériel
--   octets 0..2 = oui
--   octets 3..4 = manufacturerId
--   octets 5..8 = productId
--   octet  9    = modLevel

ResetCause::= INTEGER {
  unknown  (1),
  powerUp  (2),
  managed  (3),
  watchdog (4)
} (unknown..watchdog)
```

```
-- Énumération identifiant la cause de la dernière réinitialisation de
              l'unité.

ResetType::= INTEGER {
  hard     (1),
  soft     (2),
  shutdown (3)
} (hard..shutdown)
-- Énumération identifiant une opération de réinitialisation ou d'arrêt.

PowerType::= INTEGER {
  ac     (1), -- alimentation en courant alternatif externe
  dc     (2), -- alimentation en courant continu externe
  stored (3)    -- batterie interne ou autre dispositif de stockage de
              charge
} (ac..stored)
-- Énumération identifiant un type de source d'alimentation.

PowerStatus::= INTEGER {
  charged     (1),
  charging    (2),
  discharging (3),
  discharged  (4),
  faulty      (5),
  expired     (6)
} (charged..expired)
-- Énumération identifiant l'état d'une source d'alimentation.

ChargeLevel::= INTEGER (0..100)
-- Valeur représentant le niveau de charge d'une batterie ou d'un autre
              dispositif
-- de stockage de charge, sous forme de pourcentage.

UnitAlarms::= OCTET STRING (SIZE(1))
-- ensemble de bits représentant les alarmes générales pour une unité
--   bit 1 (lsb) = alarme de la source d'alimentation courante
--   bit 2       = autre  alarme de la source d'alimentation
--   bit 3       = alarme du serveur de temps 1
--   bit 4       = alarme du serveur de temps 2
--   bit 5       = alarme de l'horloge de référence
--   bit 6       = réservé
--   bit 7       = réservé
--   bit 8 (msb) = réservé

DateTime::= OCTET STRING (SIZE(9))
-- Valeur de code représentant une date et une heure
--   octets 0..1 = année
--   octet  2    = mois       (1..12)
--   octet  3    = jour       (1..31)
--   octet  4    = heure      (0..23)
--   octet  5    = minute     (0..59)
--   octet 6 = seconde     (0..60)  (autorise des secondes intercalaires)
--   octets 7..8 = milliseconde (0..999)

ServerNumber::= INTEGER (1..2)
-- Valeur identifiant un serveur parmi deux possibles, utilisé pour la
              synchronisation
-- d'une horloge interne en temps réel.

ServerType::= INTEGER {
  none    (1),
  private (2),
  network (3)
```

```
} (none..network)
-- Énumération identifiant le type de serveur utilisé pour
-- synchroniser une horloge interne en temps réel.

ClockSource::= INTEGER (1..255)
--  Pointeur  identifiant  de  manière  unique  une  source  d'horloge  de
            référence pour une unité.
-- La sémantique est spécifique au matériel.

SoftwareArea::= INTEGER (1..2147483647)
-- Pointeur identifiant de manière unique une zone de téléchargement de
            logiciel dans l'unité.

SoftwareClass::= INTEGER {
  general  (1),
  firmware (2)
} (1..255)
--  Énumération  identifiant  la  classe  de  stockage  d'une  zone  de
            téléchargement
-- de logiciel. La sémantique est spécifique au matériel.

SoftwareAccess::= INTEGER {
  read  (1),
  write (2),
  erase (3),
  full  (4)
} (read..full)
-- Énumération identifiant les permissions d'accès pour une zone de
-- téléchargement de logiciel.

SoftwareStatus::= INTEGER {
  empty        (1),
  erasing      (2),
  invalid      (3),
  writing      (4),
  beingWritten (5),
  valid        (6)
} (empty..valid)
-- Énumération identifiant l'état courant d'une zone de téléchargement
-- de logiciel.

SoftwareType::= INTEGER (1..2147483647)
-- Pointeur identifiant de manière unique un type de contenu de logiciel.
            La sémantique est
-- spécifique au matériel.

SoftwareSerial::= INTEGER {
  none (16)
} (0..15 | none)
--  Pointeur  utilisé  pour  identifier  le  contenu  téléchargé  le  plus
            récemment pour
-- une zone de téléchargement de logiciel.

SoftwareVersion::= OCTET STRING (SIZE(0..80))
-- Valeur de code identifiant la version de logiciel contenue dans une
-- zone de téléchargement de logiciel.

SoftwareLength::= INTEGER (1..256)
-- Valeur représentant la longueur d'une Séquence de contenus de logiciel
-- dans une zone de téléchargement de logiciel.

SoftwareContents::= OCTET STRING (SIZE(0..256))
```

```
-- Séquence de contenus de logiciel dans une zone de téléchargement de
              logiciel.

OperationId::= OCTET STRING (SIZE(10))
-- Pointeur identifiant de manière unique une opération programmée dans
              l'unité.

OperationType::= INTEGER {
  modify  (1),
  monitor (2)
} (modify..monitor)
-- Énumération représentant l'action exécutée par une opération
-- programmée.

ObjectName::= OBJECT IDENTIFIER
-- Identifiant d'objet identifiant tout objet de la MIB mis en œuvre par
-- l'unité.

ObjectValue::= OCTET STRING (SIZE(1..65535))
-- Valeur de code représentant la valeur de tout objet de la MIB mis en
              œuvre par
-- l'unité. Codée en utilisant les règles de codage de base ASN.1 (BER).

OperationState::= INTEGER {
  pending (1),
  delayed (2),
  aborted (3)
} (pending..aborted)
-- Énumération représentant l'état courant d'une opération
-- programmée.
```

4.2.3 Définitions des types de rangées conceptuelles

Les types suivants sont utilisés pour spécifier la syntaxe des objets gérés par la présente norme, représentant des rangées conceptuelles de tableaux.

```
UnitPowerSourceEntry::= SEQUENCE {
  psNumber       IndexNumber,
  psType         PowerType,
  psStatus       PowerStatus,
  psChargeLevel  ChargeLevel,
  psChargeTime   CardinalNumber
}

BlockEntry::= SEQUENCE {
  blockId    BlockId,
  blockType  BlockType
}

ConnectorEntry::= SEQUENCE {
  connRxBlockId      BlockId,
  connRxBlockInput   IndexNumber,
  connTxBlockId      BlockId,
  connTxBlockOutput  IndexNumber
}

ModeEntry::= SEQUENCE {
  mBlockId       BlockId,
  mBlockOutput   IndexNumber,
  mMediaFormat   MediaFormat,
  mEnabled       TruthValue
}
```

```
  TimeServerEntry::= SEQUENCE {
    tsNumber       ServerNumber,
    tsType         ServerType,
    tsAddressType  TDomain,
    tsAddressValue TAddress,
    tsConnectTime  CardinalNumber,
    tsLockedTime   CardinalNumber,
    tsDifference   IntegerNumber
  }

  ClockOutputEntry::= SEQUENCE {
    coNumber     IndexNumber,
    coName       Utf8String,
    coFrequency  CardinalNumber
  }

  SoftwareAreaEntry::= SEQUENCE {
    swaId       SoftwareArea,
    swaClass    SoftwareClass,
    swaAccess   SoftwareAccess,
    swaStatus   SoftwareStatus,
    swaLength   CardinalNumber,
    swaType     SoftwareType,
    swaSerial   SoftwareSerial,
    swaVersion  SoftwareVersion
  }

  SoftwareContentEntry::= SEQUENCE {
    swcAreaId  SoftwareArea,
    swcOffset  CardinalNumber,
    swcLength  SoftwareLength,
    swcData    SoftwareContents
  }

  ScheduledOpEntry::= SEQUENCE {
    soRequestId    OperationId,
    soRelativeId   OperationId,
    soType         OperationType,
    soPersistent   TruthValue,
    soDateTime     DateTime,
    soObjectName   ObjectName,
    soObjectValue  ObjectValue,
    soDescription  Utf8String,
    soPrivilege    PrivilegeLevel,
    soState        OperationState
}
```


4.2.4 Objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») pour identité d'unité de base et informations d'état

Le groupe d'objets du Tableau 1 doit être mis en œuvre par tout matériel conforme. Le nœud racine pour ces objets doit être:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) unit-information(1) }

Tableau 1 – Objets gérés acheminant des informations concernant l'unité

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
unitName(1)	Utf8String		listener	supervisor	no	m
unitLocation(2)	Utf8String		listener	supervisor	no	m
unitAddress(3)	MacAddress		listener	none	no	m
unitIdentity(4)	UnitIdentity		listener	none	no	m
unitManufacturerName(5)	Utf8String		listener	none	no	m
unitProductName(6)	Utf8String		listener	none	no	m
unitSerialNumber(7)	Utf8String		listener	none	no	m
unitFirmwareVersion(8)	Utf8String		listener	none	no	o
unitUpTime(9)	CardinalNumber		listener	none	yes	m
unitResetCause(10)	ResetCause		listener	none	no	m
unitReset(11)	ResetType		none	supervisor	yes	o
unitPowerSource(12)	IndexNumber		listener	supervisor	maybe	m
unitPowerSourceTable(13)	SEQUENCE OF UnitPowerSourceEntry		none	none	no	m
└unitPowerSourceEntry(1)	UnitPowerSourceEntry		none	none	no	m
├psNumber(1)	IndexNumber	yes	none	none	no	m
├psType(2)	PowerType		listener	none	no	m
├psStatus(3)	PowerStatus		listener	none	yes	m
├psChargeLevel(4)	ChargeLevel		listener	none	yes	o
└psChargeTime(5)	CardinalNumber		listener	none	yes	o
unitAlarmsEnabled(14)	UnitAlarms		listener	supervisor	no	o
unitAlarmsRaised(15)	UnitAlarms		listener	none	yes	o

4.2.4.1 unitName

Nom assigné à l'unité. Il s'agit d'une chaîne textuelle arbitraire assignée par le gestionnaire du système.

NOTE Si une unité met également en œuvre l'objet de la MIB 1.3.6.1.2.1.1.5.0 (`sysName`) défini dans la RFC1213, il convient que `unitName` soit un alias pour cet objet.

4.2.4.2 unitLocation

Emplacement physique de l'unité. Il s'agit d'une chaîne textuelle arbitraire assignée par le gestionnaire du système.

NOTE Si une unité met également en œuvre l'objet de la MIB 1.3.6.1.2.1.1.6.0 (`sysLocation`) défini dans la RFC1213, il convient que `unitLocation` soit un alias pour cet objet.

4.2.4.3 unitAddress

Adresse de l'unité sur 48 bits. Celle-ci doit être une adresse MAC individuelle de LAN, comme spécifié en 9.2 de l'IEEE Std 802 (et non une adresse de multidiffusion) et il convient qu'elle soit l'adresse globalement unique de l'une de ses interfaces. Si toutefois une unité n'a pas d'adresse globalement unique, une adresse locale devant être unique dans le réseau doit être utilisée. La façon dont cette adresse locale est allouée ne fait pas partie du domaine d'application de la présente norme.

4.2.4.4 unitIdentity

Collection d'informations identifiant de manière unique le produit dont l'unité est un exemple.

- `oui` est le OUI (Identificateur unique d'organisation, en anglais « Organizationally unique identifier ») utilisé par le fabricant de produit pour ce produit;
- `manufacturerId` est l'identifiant unique assigné au fabricant par le propriétaire du OUI;
- `productId` est l'identifiant unique assigné au produit par le fabricant; et
- `modLevel` identifie toutes les modifications effectuées à la conception du produit ou à l'unité après qu'elle a été fabriquée.

4.2.4.5 unitManufacturerName

Nom du fabricant de l'unité. Il s'agit d'une chaîne textuelle arbitraire assignée par le fabricant de l'unité.

4.2.4.6 unitProductName

Nom de produit assigné à l'unité par son fabricant. Il s'agit d'une chaîne textuelle arbitraire assignée par le fabricant de l'unité.

4.2.4.7 unitSerialNumber

Numéro de série assigné à l'unité par son fabricant. Il s'agit d'une chaîne textuelle arbitraire assignée par le fabricant de l'unité.

4.2.4.8 unitFirmwareVersion

Numéro de version du progiciel installé dans l'unité. Il s'agit d'une chaîne textuelle arbitraire assignée par le fabricant de l'unité.

Cet objet doit être obligatoire si un élément matériel ne prend pas en charge la mise à jour du progiciel par l'intermédiaire du groupe d'objets spécifié en 4.2.8.

4.2.4.9 unitUpTime

Nombre de secondes depuis que l'unité a été mise sous tension ou réinitialisée pour la dernière fois.

4.2.4.10 unitResetCause

Cause de la dernière réinitialisation de l'unité.

4.2.4.11 unitReset

Provoque la réinitialisation ou l'arrêt de l'unité (comme spécifié par la valeur fixée). Après une réinitialisation ou un arrêt initialisé par l'écriture dans cet objet, la cause de la réinitialisation doit être fixée à `managed`.

4.2.4.12 unitPowerSource

Numéro d'index de l'entrée dans le tableau de sources d'alimentation de l'unité représentant la source courante d'alimentation de l'unité. Ne peut être écrit que si l'unité permet une commutation manuelle entre sources d'alimentation.

4.2.4.13 unitPowerSourceTable

Tableau de descripteurs de source d'alimentation. Chaque source d'alimentation de l'unité possède une entrée dans ce tableau.

4.2.4.14 unitPowerSourceEntry

Entrée dans le tableau de sources d'alimentation de l'unité.

4.2.4.15 psNumber

Numéro de source d'alimentation. Utilisé comme index lors de l'accès au tableau de sources d'alimentation de l'unité. Chaque source d'alimentation dans une unité possède un numéro unique.

NOTE Il convient d'allouer en séquence les numéros de source d'alimentation en partant de 1.

4.2.4.16 psType

Type de source d'alimentation.

4.2.4.17 psStatus

Etat courant de cette source d'alimentation.

- `charged` indique que la source d'alimentation est entièrement chargée. Pour une alimentation externe, ceci est utilisé pour indiquer que la tension d'alimentation est supérieure au seuil exigé par l'unité.
- `charging` indique que la source d'alimentation est une batterie (ou un autre dispositif stockant une charge) qui se recharge elle-même à partir d'une autre source d'alimentation mais qui n'est pas encore entièrement chargée.
- `discharging` indique que la source d'alimentation est une batterie (ou un autre dispositif stockant une charge) qui ne se recharge pas elle-même à partir d'une autre source d'alimentation et qui n'est pas entièrement chargée ou entièrement déchargée.
- `discharged` indique que la source d'alimentation est entièrement déchargée. Pour une alimentation externe, ceci est utilisé pour indiquer que la tension d'alimentation est inférieure au seuil exigé par l'unité.
- `faulty` indique qu'un défaut de la source d'alimentation a été détecté.
- `expired` indique que la source d'alimentation est une batterie (ou un autre dispositif stockant une charge) devant être remplacée.

4.2.4.18 psChargeLevel

Niveau de charge courant de cette source d'alimentation sous forme du pourcentage de pleine charge.

4.2.4.19 psChargeTime

Si l'état courant de cette source d'alimentation est `charging`, ceci indique le temps estimé (en minutes) jusqu'à ce que la source d'alimentation soit entièrement chargée. Si l'état courant de cette source d'alimentation est `discharging`, ceci indique le temps estimé (en minutes) jusqu'à ce que la source d'alimentation soit entièrement déchargée. Si l'état courant de cette source d'alimentation est une quelconque autre valeur, celui-ci a pour valeur `0`.

4.2.4.20 unitAlarmsEnabled

Indique les alarmes qui sont activées pour cette unité. Les bits représentant des alarmes qui ne sont pas mises en œuvre par l'unité doivent toujours être lus comme des zéros.

4.2.4.21 unitAlarmsRaised

Indique les alarmes qui sont déclenchées pour cette unité.

4.2.5 Objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») pour le cadre de bloc

Le groupe d'objets du Tableau 2 doit être mis en œuvre par tout matériel conforme. Le nœud racine pour ces objets doit être

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) block-framework(2) }

Tableau 2 – Objets gérés pour la configuration de bloc et de connecteur

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
blockTable(1)	SEQUENCE OF BlockEntry		none	none	no	m
└blockEntry(1)	BlockEntry		none	none	no	m
├blockId(1)	BlockId	yes	none	none	no	m
└blockType(2)	BlockType		listener	supervisor	no	m
connectorTable(2)	SEQUENCE OF ConnectorEntry		none	none	no	m
└connectorEntry(1)	ConnectorEntry		none	none	no	m
├connRxBlockId(1)	BlockId	yes	none	none	no	m
├connRxBlockInput(2)	IndexNumber	yes	none	none	no	m
├connTxBlockId(3)	SourceBlockId		listener	supervisor	no	m
└connTxBlockOutput(4)	IndexNumber		listener	supervisor	no	m
modeTable(3)	SEQUENCE OF ModeEntry		none	none	no	m
└modeEntry(1)	ModeEntry		none	none	no	m
├mBlockId(1)	BlockId	yes	none	none	no	m
├mBlockOuput(2)	IndexNumber	yes	none	none	no	m
├mMediaFormat(3)	MediaFormat	yes	none	none	no	m
└mEnabled(4)	TruthValue		listener	supervisor	no	m

4.2.5.1 blockTable

Tableau de descripteurs de bloc. Chaque bloc de l'unité possède une entrée dans ce tableau.

4.2.5.2 blockEntry

Entrée dans le tableau de blocs.

4.2.5.3 blockId

Identifiant de bloc. Utilisé comme index lors de l'accès au tableau de blocs. Chaque bloc d'une unité possède un identifiant unique indépendant du type de bloc.

4.2.5.4 blockType

Le type de bloc identifie le nœud dans l'arborescence d'identifiants d'objet qui constitue la racine de tous les objets gérés utilisés pour commander les blocs de ce type. Ne peut être écrit que si l'unité permet la reconfiguration des blocs. L'écriture d'une valeur NULL doit demander la suppression du bloc. L'écriture d'une valeur autre que NULL pour un `blockId` qui n'existe pas doit demander la création d'un nouveau bloc dont aucune entrée n'est reliée.

4.2.5.5 connectorTable

Tableau de descripteurs de connecteurs. Chaque connecteur de l'unité possède une entrée dans ce tableau.

4.2.5.6 connectorEntry

Entrée dans le tableau de connecteurs.

4.2.5.7 connRxBlockId

Identifiant de bloc du bloc constituant l'extrémité de réception de la liaison. Utilisé comme index lors de l'accès au tableau de connecteurs.

4.2.5.8 connRxBlockInput

Numéro d'entrée de l'entrée de bloc constituant l'extrémité de réception de la liaison. Utilisé comme index lors de l'accès au tableau de connecteurs.

4.2.5.9 connTxBlockId

Identifiant de bloc du bloc constituant l'extrémité de transmission de la liaison. Ne peut être écrit que si l'unité permet la reconfiguration des liaisons. La valeur `nullBlock` indique que l'entrée n'est pas reliée.

4.2.5.10 connTxBlockOutput

Numéro de sortie de la sortie du bloc constituant l'extrémité de transmission de la liaison. Ne peut être écrit que si l'unité permet la reconfiguration des liaisons.

4.2.5.11 modeTable

Tableau de descripteurs de modes. Pour chaque sortie de bloc dans l'unité il existe une ou plusieurs entrées dans ce tableau, spécifiant les formats de données valables pour cette sortie et commandant si la sortie est actuellement autorisée à fonctionner dans ce mode.

NOTE Le tableau de mode peut être utilisé pour les objectifs suivants:

- permettre à un logiciel de commande générique de déterminer les capacités d'une unité particulière;
- permettre une commande manuelle de la sortie de format de données par un bloc particulier;
- permettre d'imposer des restrictions à un bloc qui sélectionne automatiquement son format de données de sortie.

Lorsqu'il est utilisé pour la commande manuelle du format de données de sortie, il convient que l'activation d'un mode désactive automatiquement tous les autres modes pour cette sortie.

4.2.5.12 modeEntry

Entrée dans le tableau de modes.

4.2.5.13 mBlockId

Identifiant de bloc. Utilisé comme index lors de l'accès au tableau de modes.

4.2.5.14 mBlockOutput

Numéro de sortie de bloc. Utilisé comme index lors de l'accès au tableau de modes.

4.2.5.15 mMediaFormat

Format de support associé à ce mode. Utilisé comme index lors de l'accès au tableau de modes.

4.2.5.16 mEnabled

S'il est à vrai, indique que la sortie de bloc associée peut fonctionner dans ce mode.

4.2.6 Objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») pour les informations d'horloge en temps réel

Le groupe d'objets du Tableau 3 doit être mis en œuvre par toute unité contenant une horloge interne en temps réel. Le nœud racine pour ces objets doit être

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) time(3) }

Tableau 3 – Objets de la MIB pour les informations d'horloge en temps réel

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
timeOfDay(1)	DateTime		listener	supervisor	no	m
timeZone(2)	INTEGER (-720..840)		listener	supervisor	no	m
timeAdvance(3)	INTEGER (0..120)		listener	supervisor	no	m
timeErrorThreshold(4)	INTEGER (0..250)		listener	supervisor	no	o
timeAlarmDelay(5)	INTEGER (0..240)		listener	supervisor	no	o
timeServerTable(6)	SEQUENCE OF TimeServerEntry		none	none	no	o
└timeServerEntry(1)	TimeServerEntry		none	none	no	m
├tsNumber(1)	ServerNumber	yes	none	none	no	m
├tsType(2)	ServerType		listener	supervisor	no	m
├tsAddressType(3)	TDomain		listener	supervisor	no	o
├tsAddressValue(4)	TAddress		listener	supervisor	no	o
├tsConnectTime(5)	CardinalNumber		listener	none	yes	m
├tsLockedTime(6)	CardinalNumber		listener	supervisor	yes	m
└tsDifference(7)	IntegerNumber		listener	none	yes	m

4.2.6.1 timeOfDay

Date et heure UTC selon l'horloge interne de l'unité.

4.2.6.2 timeZone

Différence (en minutes) entre l'heure locale et l'heure UTC, à l'exclusion de tout réglage effectué pour l'heure d'été.

4.2.6.3 timeAdvance

Valeur (en minutes) dont l'heure locale a été avancée pour l'heure d'été.

4.2.6.4 timeErrorThreshold

Valeur (en millisecondes) dont deux sources horaires sont autorisées à différer avant qu'elles ne soient considérées comme étant en désaccord.

4.2.6.5 timeAlarmDelay

Temps (en secondes) pendant lequel une source horaire doit être en désaccord avant de déclencher une alarme.

4.2.6.6 timeServerTable

Tableau de descripteurs de serveurs de temps pour cette unité. Il doit y avoir deux entrées dans ce tableau.

4.2.6.7 timeServerEntry

Entrée dans le tableau de serveurs de temps.

4.2.6.8 tsNumber

Numéro de serveur de temps. Utilisé comme index lors de l'accès au tableau de serveurs de temps.

4.2.6.9 tsType

Type de serveur de temps.

4.2.6.10 tsAddressType

Type d'adresse réseau utilisée pour localiser ce serveur de temps et s'y raccorder. Ceci définit la sémantique associée à `tsAddressValue`.

NOTE Si l'unité ne met pas en œuvre cet objet, l'interprétation de `tsAddressValue` nécessite des informations acheminées autrement que par la présente norme, par exemple, telles que le système ne prend en charge qu'une seule sorte d'adresse. Les matériels qui ne prennent en charge qu'une sorte de type d'adresse peuvent mettre en œuvre cet objet en lecture seule, et il peut être présent même si le type de serveur n'est pas « network ».

4.2.6.11 tsAddressValue

Adresse réseau utilisée pour localiser ce serveur de temps et s'y raccorder. Utilisé uniquement si le type de serveur de temps est fixé à « network ». Si l'unité met en œuvre `tsAddressType`, la valeur d'adresse doit être interprétée en fonction de la valeur courante de `tsAddressType`.

4.2.6.12 tsConnectTime

Temps (en secondes) pendant lequel l'unité a maintenu une liaison ininterrompue avec ce serveur de temps.

4.2.6.13 tsLockedTime

Temps (en secondes) pendant lequel l'horloge interne de l'unité a été en accord avec l'heure du jour diffusée par ce serveur de temps.

4.2.6.14 tsDifference

Différence (en millisecondes) entre l'horloge interne de l'unité et l'heure du jour diffusée par ce serveur de temps. Une différence positive indique que l'heure de l'unité est en avance, une différence négative indique qu'elle est en retard.

4.2.7 Objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») pour les informations d'horloge de référence

Le groupe d'objets du Tableau 4 doit être mis en œuvre par toute unité utilisant ou fournissant en sortie une horloge de référence. Le nœud racine pour ces objets doit être:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) clock(4) }

Tableau 4 – Objets gérés pour les informations d'horloge de référence

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
clockSource(1)	ClockSource		listener	none	yes	m
clockFrequency 2)	CardinalNumber		listener	none	no	m
clockLockedTime(3)	CardinalNumber		listener	none	no	m
clockAlarmDelay(4)	CardinalNumber		listener	supervisor	no	o
clockOutputTable(5)	SEQUENCE OF ClockOutputEntry		none	none	no	o
└clockOutputEntry(1)	ClockOutputEntry		none	none	no	m
├coNumber(1)	IndexNumber	yes	none	none	no	m
├coName(2)	Utf8String		listener	supervisor	no	m
└coFrequency(3)	CardinalNumber		listener	supervisor	no	m

4.2.7.1 clockSource

Source du signal d'horloge de référence.

4.2.7.2 clockFrequency

Fréquence nominale (en Hz) du signal d'horloge de référence. Une valeur de zéro indique qu'aucun signal n'est présent ou que la fréquence ne peut pas être mesurée.

4.2.7.3 clockLockedTime

Temps (en secondes) pendant lequel l'unité a été verrouillée sur le signal d'horloge de référence.

4.2.7.4 clockAlarmDelay

Temps (en secondes) pendant lequel l'horloge de référence doit être absente ou en dehors de la plage avant de déclencher une alarme.

4.2.7.5 clockOutputTable

Tableau de descripteurs de sorties d'horloge pour cette unité.

4.2.7.6 clockOutputEntry

Entrée dans le tableau de sorties d'horloge.

4.2.7.7 coNumber

Numéro de sortie. Utilisé comme index lors de l'accès au tableau de sorties d'horloge.

4.2.7.8 coName

Non assigné à la sortie d'horloge. Il s'agit d'une chaîne textuelle arbitraire assignée par le gestionnaire du système.

4.2.7.9 coFrequency

Fréquence d'horloge nominale requise (en Hz) pour cette sortie.

4.2.8 Objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») pour téléchargement de logiciel

Le groupe d'objets du Tableau 5 doit être mis en œuvre par tous les matériels prenant en charge le téléchargement à distance de logiciel ou d'autres fichiers de configuration. Le nœud racine pour ces objets doit être:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) software(5) }

Tableau 5 – Objets gérés pour téléchargement de logiciel

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
softwareAreaTable(1)	SEQUENCE OF SoftwareAreaEntry		none	none	no	m
└softwareAreaEntry(1)	SoftwareAreaEntry		none	none	no	m
├swaId(1)	SoftwareArea	yes	none	none	no	m
├swaClass(2)	SoftwareClass		maintenance	none	no	m
├swaAccess(3)	SoftwareAccess		maintenance	none	no	m
├swaStatus(4)	SoftwareStatus		maintenance	maintenance	no	m
├swaLength(5)	CardinalNumber		maintenance	maintenance	no	m
├swaType(6)	SoftwareType		maintenance	maintenance	no	m
├swaSerial(7)	SoftwareSerial		maintenance	maintenance	no	m
└swaVersion(8)	SoftwareVersion		maintenance	none	no	m
softwareContentTable(2)	SEQUENCE OF SoftwareContentEntry		none	none	no	m
└softwareContentEntry(1)	SoftwareContentEntry		none	none	no	m
├swcAreaId(1)	SoftwareArea	yes	none	none	no	m
├swcOffset(2)	CardinalNumber	yes	none	none	no	m
├swcLength(3)	SoftwareLength	yes	none	none	no	m
└swcData(4)	SoftwareContents		maintenance	maintenance	no	m

4.2.8.1 softwareAreaTable

Tableau de descripteurs de zone de téléchargement de logiciel pour cette unité.

4.2.8.2 softwareAreaEntry

Entrée dans le tableau de zones de téléchargement de logiciel.

4.2.8.3 swaId

Pointeur identifiant de manière unique une zone de téléchargement de logiciel dans l'unité. Utilisé comme index lors de l'accès au tableau de zones de téléchargement de logiciel.

4.2.8.4 swaClass

Classe de stockage de cette zone de téléchargement de logiciel.

4.2.8.5 swaAccess

Accessibilité de cette zone de téléchargement de logiciel.

4.2.8.6 swaStatus

Etat courant de cette zone de téléchargement de logiciel. Un état

- `empty` signifie que la zone a été effacée (le cas échéant) et ne contient aucune donnée;
- `erasing` signifie qu'une demande d'effacement de toutes les données contenues dans la zone a été acceptée mais que le processus n'est pas encore terminé;
- `invalid` signifie que la zone n'est pas vide mais que son format n'est conforme à aucun des types pris en charge par l'unité;
- `writing` signifie que le terminal de gestion auquel RESPONSE est envoyé peut modifier le contenu;
- `beingWritten` signifie qu'une autre entité (un autre terminal de gestion ou l'unité gérée) peut modifier le contenu; et
- `valid` signifie que la zone contient des données dans un format pris en charge par l'unité.

4.2.8.7 swaLength

Nombre d'octets dans cette zone de téléchargement de logiciel.

4.2.8.8 swaType

Type de logiciel contenu dans cette zone de téléchargement de logiciel.

4.2.8.9 swaSerial

Numéro de série assigné au dernier téléchargement terminé de cette zone de téléchargement de logiciel.

4.2.8.10 swaVersion

Numéro de version codé du contenu de cette zone de téléchargement de logiciel. Le numéro de version est déterminé d'après le contenu de la zone d'une manière qui dépend du type et qui n'est pas ici défini; sa longueur est nulle si le contenu ne comporte pas de numéro de version.

4.2.8.11 softwareContentTable

Tableau de descripteurs de contenu de téléchargement de logiciel pour cette unité. Ce tableau comporte une entrée théorique pour chaque séquence contiguë possible d'octets dans chaque zone de téléchargement de logiciel.

4.2.8.12 softwareContentEntry

Entrée dans le tableau de contenus de téléchargement de logiciel.

4.2.8.13 swcAreaId

Pointeur identifiant de manière unique une zone de téléchargement de logiciel dans l'unité. Utilisé comme index lors de l'accès au tableau de contenus de téléchargement de logiciel.

4.2.8.14 swcOffset

Décalage (en octets) de cette séquence de contenus par rapport au début de la zone de téléchargement de logiciel spécifiée. Utilisé comme index lors de l'accès au tableau de contenus de téléchargement de logiciel.

4.2.8.15 swcLength

Longueur (en octets) de cette séquence de contenus. Utilisé comme index lors de l'accès au tableau de contenus de téléchargement de logiciel.

4.2.8.16 swcData

Valeur de cette séquence de contenus.

4.2.9 Objets de la MIB (Base d'informations de gestion, en anglais « Management information base ») pour les opérations programmées

Le groupe d'objets du Tableau 6 doit être mis en œuvre par tout matériel prenant en charge des opérations programmées. Le nœud racine pour ces objets doit être:

{ iso(1) standard(0) IEC62379(62379) general(1) generalMIB(1) schedule(6) }

Tableau 6 – Objets gérés pour des opérations programmées

Identifier	Syntax	Index	Readable	Writable	Volatile	Status
scheduledOpTable(1)	SEQUENCE OF ScheduledOpEntry		none	none	no	m
└scheduledOpEntry(1)	ScheduledOpEntry		none	none	no	m
├soRequestId(1)	OperationId	yes	none	none	no	m
├soRelativeId(2)	OperationId		listener	operator	no	m
├soType(3)	OperationType		listener	operator	no	m
├soPersistent(4)	TruthValue		listener	operator	no	m
├soDateTime(5)	DateTime		listener	operator	no	m
├soObjectName(6)	ObjectName		listener	operator	no	m
├soObjectValue(7)	ObjectValue		listener	operator	no	m
├soDescription(8)	Utf8String		listener	operator	no	m
├soPrivilege(9)	PrivilegeLevel		listener	operator	no	m
└soState(10)	OperationState		listener	operator	no	m

4.2.9.1 scheduledOpTable

Tableau de descripteurs d'opérations programmées pour cette unité. Le nombre d'entrées dans le tableau est déterminé par le nombre d'opérations en instance.

4.2.9.2 scheduledOpEntry

Entrée dans le tableau d'opérations programmées.

4.2.9.3 soRequestId

Identifiant de requête pour cette opération. Il s'agit d'une valeur de chaîne d'octets unique arbitraire assignée par l'unité de gestion effectuant la requête. Utilisé comme index lors de l'accès à la liste d'opérations programmées.

NOTE Il est simplement nécessaire que l'identifiant de requête soit unique par rapport aux autres opérations programmées sur la même unité.

4.2.9.4 soRelativeId

Identifiant d'une opération programmée précédemment que cette opération doit suivre. Si elle est fixée à tout à zéro, il n'est pas nécessaire que cette opération suive une quelconque autre opération.

NOTE Une opération devant suivre une opération programmée précédemment est désormais appelée opération relative, tandis qu'une opération dont il n'est pas exigé qu'elle suive une opération programmée précédemment est appelée opération absolue.

4.2.9.5 soType

Type de cette opération. Commande la façon dont l'unité gère cette opération.

NOTE Une opération avec une valeur `soType` à modify est désormais appelée opération de modification et une opération avec une valeur `soType` à monitor est désormais appelée opération de moniteur.

4.2.9.6 soPersistent

Indique si cette opération est permanente ou transitoire. Si `soPersistent` est mis à true, l'opération reste dans le tableau d'opérations programmées après avoir été exécutée, sinon, elle est enlevée du tableau d'opérations programmées après avoir été exécutée.

NOTE Une opération avec une valeur `soPersistent` à `true` est désormais appelée opération persistante et une opération avec une valeur `soPersistent` à `false` est désormais appelée opération transitoire.

4.2.9.7 soDateTime

Pour une opération absolue, heure à laquelle l'opération est programmée pour être exécutée. Pour une opération relative, heure à laquelle l'opération est programmée pour être exécutée par rapport à l'heure réelle à laquelle l'exécution de l'opération spécifiée par `soRelativeId` a été achevée.

NOTE Si une opération est retardée, toute opération associée est ainsi retardée de la même valeur.

4.2.9.8 soObjectName

Pour une opération de surveillance, instance d'objet de la MIB (Base d'informations de gestion, en anglais « Management information base ») que cette opération doit surveiller, sinon, instance d'objet de la MIB que cette opération doit modifier.

4.2.9.9 soObjectValue

Pour une opération de surveillance, valeur de déclenchement pour l'instance d'objet spécifiée par `soObjectName`; sinon, nouvelle valeur de l'instance d'objet spécifiée par `soObjectName`.

NOTE Il faut que la valeur de l'objet soit fournie sous forme d'une chaîne d'octets représentant la valeur réelle codée de BER (Règles de codage de base, en anglais « Basic encoding rules ») ASN.1. Ceci est destiné à assurer la compatibilité avec SMIv2, qui n'autorise pas de types de variant pour un objet géré.

4.2.9.10 soDescription

Chaîne textuelle arbitraire décrivant cette opération.

4.2.9.11 soPrivilege

Niveau de privilège pour cette opération.

4.2.9.12 soState

Etat courant de cette opération.

5 Procédures

5.1 Horloges en temps réel

Toute unité contenant une horloge interne en temps réel qui n'est pas verrouillée par ailleurs sur une horloge de référence externe doit essayer de surveiller en continu l'heure des diffusions quotidiennes de deux serveurs de temps reliés au réseau. Lorsqu'une liaison est effectuée avec l'un de ces serveurs, une horloge de référence locale doit être créée, verrouillée sur l'heure diffusée par ce serveur, avec un filtrage approprié pour atténuer toute gigue introduite par le réseau.

Lorsqu'on ne peut pas effectuer de liaison avec un serveur de temps, l'horloge interne de l'unité doit fonctionner librement. Lorsque l'on ne peut effectuer de liaison qu'avec un seul serveur, l'unité doit utiliser l'horloge de référence déterminée d'après ce serveur pour corriger son horloge interne. Lorsqu'on peut effectuer une liaison avec les deux serveurs, l'unité doit vérifier si les horloges de référence déterminées d'après les deux serveurs sont maintenues à l'intérieur du seuil d'erreur de temps fixé pour l'unité. Dans l'affirmative, elle doit utiliser la moyenne des deux horloges de référence pour corriger son horloge interne. Sinon, elle doit utiliser l'horloge de référence la plus proche de son horloge interne pour corriger son horloge interne.

5.2 Procédures de téléchargement de logiciel

5.2.1 Zones

Le terminal de gestion doit demander la suppression d'une zone en fixant son status à « effacement », l'unité gérée doit alors supprimer son contenu courant et régler le status à « vide ». Il peut ensuite l'amalgamer avec une autre zone vide.

Le terminal de gestion doit demander la permission d'écrire dans une zone en fixant son status à « writing »; l'unité gérée doit rejeter la demande si le status précédent était différent de « empty » ou « valid ». Si la demande réussit, elle verra l'état « writing », mais tous les autres terminaux de gestion le verront comme « being written ». Le type ne doit pas être modifié, sauf après avoir modifié l'état de « empty » à « writing » et avant d'écrire quoi que ce soit dans le contenu (voir 5.2.2). Lorsque l'ensemble du contenu a été écrit, le terminal de gestion doit fixer `status` à « valid »; il peut également être indiqué dans `RESPONSE` comme « written » si l'unité gérée a besoin de temps pour écrire le type, etc., dans la mémoire.

Si l'unité gérée est réinitialisée ou si la liaison de gestion est annulée (dans le cas d'un service de transfert de paquets orienté connexion) ou si la temporisation est atteinte (dans le cas d'un service de transfert de paquets sans connexion), pendant que status est à « writing », alors si le status précédent était « valid », il doit revenir à « valid », sinon il doit passer à « invalid ». L'unité gérée peut effacer les zones non valables, en les convertissant en « empty ».

Des valeurs spécifiques au produit pour `class` et `type` doivent être définies par le fabricant du produit; il convient qu'il n'existe qu'un seul ensemble de valeurs de type pour un produit donné quelconque, et non un par classe.

NOTE 1 Il n'est pas nécessaire que le terminal de gestion comprenne la signification de la classe spécifique au produit ou les valeurs de type.

La longueur doit être la longueur de la zone; les données qu'elle contient peuvent être plus courtes.

NOTE 2 La méthode d'indication de la longueur des données n'est pas définie ici et est probablement différente pour chaque type.

Une zone avec un numéro de série *s* (dans la gamme de 0 à 15 inclus) doit être une zone « latest » de son type s'il n'y a pas de zone de ce type avec le numéro de série *s* + 1 (modulo 16).

S'il y a plus d'une zone « latest » d'un type donné ou plus d'une zone de mêmes type et numéro de série ou s'il existe des zones avec l'ensemble des 16 valeurs de numéro de série possibles pour un type donné, la mémoire de l'unité est alors « malformed ». Il convient qu'une unité gérée rejette tout message `SET` conduisant à une malformation de sa mémoire. il convient qu'un terminal de gestion découvrant qu'une mémoire de l'unité est malformée en informe l'utilisateur. Le comportement d'une unité avec une mémoire malformée et l'action pour corriger le problème sont spécifiques au produit et ne font pas partie du domaine d'application de la présente norme.

Si la mémoire de l'unité n'est pas malformée, la zone « current » de chaque type doit être sélectionnée comme suit.

- Soit *s* le numéro de série de la dernière zone du type requis.
- Le contenu de la zone choisie doit être contrôlé; si le contrôle échoue, *s* doit alors être réduit de 1 (modulo 16) et s'il existe une zone du type requis avec le numéro de série (la nouvelle valeur de) *s* cette étape doit être répétée.

NOTE 3 La présente norme ne spécifie pas les contrôles qui sont à effectuer; il est probable qu'ils comportent le calcul d'une somme de contrôle ou CRC.

- S'il existe maintenant une zone du type requis avec le numéro de série *s*, elle doit être désignée comme zone « current » de ce type, sinon, il n'y a pas de zone « current » de ce type.

Lors du chargement d'un logiciel ou d'autres informations de configuration au démarrage, l'unité doit les prendre dans la zone « current » du type approprié, devant alors être désignée comme zone « active » de ce type. Lorsque la zone « current » change (en conséquence de la procédure spécifiée en 5.2.3), l'unité peut ou non modifier la zone « active » pour correspondre.

NOTE 4 Pour un logiciel, il est habituellement approprié que la zone « active » soit la zone choisie au démarrage mais pour certaines informations de configuration, l'unité peut démarrer en utilisant immédiatement la nouvelle version, sans redémarrer. Il sera nécessaire de vérifier la zone en premier, pour s'assurer qu'elle est « current » et pas simplement « latest ».

Une zone « active » doit être en lecture seule.

NOTE 5 Ceci assure que le logiciel existant ne peut pas être effacé avant qu'un nouveau logiciel ait été chargé et soit exécuté.

5.2.2 Contenu

Un message `SET` pour une entrée dans le tableau de « contents » doit être traité comme une demande d'écriture du contenu de la zone identifiée par `handle`. L'unité gérée doit répondre par le code d'erreur « readOnly » si l'état de la zone vu par l'expéditeur du message `SET` n'est pas « writing ».

NOTE 1 La longueur est limitée par la taille de la chaîne d'octets s'ajustant dans un message de longueur maximale.

Lorsque l'unité gérée reçoit un message `SET`, elle doit écrire dans la mémoire physique, puis relire et placer le résultat dans `RESPONSE`. Un `GET` ou `SET` pour une combinaison de décalage et de longueur ne s'ajustant pas dans la longueur de la zone peut être rejeté (avec le code

d'erreur « tooBig ») ou peut être tronqué, auquel cas la longueur dans RESPONSE doit représenter la longueur réellement écrite.

NOTE 2 RESPONSE sert de confirmation non seulement que le message a été reçu mais également que les données ont été écrites correctement. Le temps nécessaire pour écrire dans la mémoire et la relire n'est probablement pas suffisamment long pour retarder RESPONSE d'une manière significative. Lors de l'écriture dans une zone, les performances seront améliorées si le terminal de gestion envoie le deuxième message SET avant d'attendre la réponse au premier, mais s'il envoie un trop grand nombre de messages SET avant d'attendre une réponse de l'unité gérée (ou du réseau), il peut en perdre une partie en raison d'un manque d'espace de tampon.

5.2.3 Procédure de mise à jour d'un composant logiciel

Chaque composant doit être associé à une classe et à un type de zone (voir 5.2.5).

La procédure par laquelle un terminal de gestion met à jour un composant d'un type donné doit être la suivante.

- Le tableau de « zones » doit d'abord être balayé pour localiser une zone vide d'une taille suffisante et de la classe requise et pour trouver le numéro de série de la dernière zone du type requis. S'il n'y a aucune zone de la classe requise, il doit rechercher en remplacement une zone vide de classe 1. Lors de la recherche de composants existants, il convient d'ignorer la « classe ». Si la mémoire de l'unité est malformée, il convient d'interrompre le processus et une intervention de l'utilisateur est nécessaire pour effacer les versions indésirables.

NOTE 1 Si on ne trouve aucune zone vide appropriée, il est possible de libérer une partie de l'espace en effaçant les versions plus anciennes du composant. Si 15 versions sont déjà présentes, il convient d'effacer la plus ancienne car sinon la mémoire se retrouvera malformée lorsque la 16ème sera écrite; il convient que le terminal de gestion ne s'attende pas à ce que l'unité gérée l'efface de sa propre initiative.

- L'état de la zone choisie doit être mis à « writing » et le type et la longueur de la zone doit ensuite être fixés comme requis pour le composant logiciel à charger. La longueur peut être arrondie par excès par l'unité gérée, par exemple (dans le cas d'une mémoire flash) à un nombre entier de blocs moins l'espace nécessaire pour contenir le type, etc. La valeur retournée dans RESPONSE doit être la longueur réelle après arrondi par excès.
- Le numéro de série peut être écrit, soit avant soit après les données; sa valeur doit être d'une unité de plus (modulo 16) que le numéro de série de la dernière zone du même type; ou toute valeur dans la plage de 0 à 15 si aucune zone de ce type n'a été trouvée.
- Le terminal de gestion doit ensuite écrire les données dans la zone. Il convient qu'il utilise une taille de fenêtre de 2 , c'est-à-dire que St doit avoir à tout moment jusqu'à 2 (mais pas plus) messages SET pour lesquels aucune RESPONSE n'a encore été reçue. Il convient ensuite qu'il démarre en envoyant deux messages SET, puis d'attendre d'avoir vu la RESPONSE au premier avant d'envoyer le troisième.
- Enfin, l'état doit être fixé à « valid ».

NOTE 2 Un unitReset sera normalement requis avant que l'unité n'exécute le nouveau logiciel.

5.2.4 Format de fichier pour un composant logiciel

Chaque composant logiciel doit recevoir un fichier dans le format suivant.

Le fichier doit commencer par deux lignes de texte codé selon l'ISO/CEI 646 (caractères sur 7 bits avec zéro comme bit supérieur) comme suit:

```
#62379.iec.ch::[format]:[domain]#[product+cpt]~[version]
length = [length][other text]
```

La première ligne doit se terminer, soit par un caractère de retour chariot simple (code $0D_{16}$) soit par une paire retour chariot-saut de ligne (codes $0D_{16}$ $0A_{16}$); la deuxième ligne doit se terminer par un caractère de retour chariot simple (code $0D_{16}$). Les parties variables sont:

[format]	`memoryimagebinary` ou `memoryimagehex`
[domain]	nom de domaine détenu par le fabricant de l'unité gérée
[product+cpt]	texte identifiant le produit et le composant
[version]	texte identifiant la version
[length]	nombre d'octets en hexadécimal (les chiffres A-F peuvent être en majuscules ou en minuscules)
[other text]	réservé pour utilisation future (par exemple, pour ajouter une somme de contrôle); il convient qu'il soit vide lorsqu'il est écrit et ignoré lorsqu'il est lu; s'il n'est pas vide, le premier caractère doit être un point-virgule («; », code $3B_{16}$)
[product+cpt]	doit être constitué de zéro ou plusieurs caractères dont les codes se trouvent dans la plage de 20_{16} à $7E_{16}$ inclus. Son format doit être spécifié par le propriétaire du *[domain]*.

[version] doit être constitué de zéro ou plusieurs caractères dont les codes se trouvent dans la plage de 20_{16} à $7D_{16}$ inclus. Son format doit être spécifié par le propriétaire du *[domain]*.

NOTE 1 Ces dispositions interdisent des caractères non imprimables autres qu'un espace; la deuxième interdit également le caractère tilde (« ~ ») dans la version, de sorte que *[product+cpt]* est constitué de tout ce qui est compris entre le premier « # » et le dernier « ~ » sur la ligne.

EXEMPLE:
```
#62379.iec.ch::memoryimagehex:ninetiles.com#RM003    logic~0.1.0    BETA    38
length = 28C44
```

Le reste du fichier doit être constitué d'une séquence de valeurs d'octet.

Si le format est `memoryimagehex`, chaque valeur d'octet doit être constituée de deux chiffres hexadécimaux (A à F pouvant être en majuscules ou en minuscules). Des caractères d'espace blanc (codes 00_{16} à 20_{16} inclus) peuvent apparaître entre des valeurs d'octets, et aussi entre le premier et après le dernier, mais pas entre deux chiffres d'un octet.

NOTE 2 Dans ce format, la totalité du fichier et du texte sauf sa taille est plus de deux fois plus grande que la taille des données à télécharger.

Si le format est `memoryimagebinary`, chaque valeur d'octet doit être directement mémorisée dans un octet du fichier, la première valeur d'octet se trouvant dans l'octet qui suit le retour chariot qui termine la deuxième ligne.

NOTE 3 Avec ce format, la taille du fichier est minimale.

Dans l'un ou l'autre format, le nombre d'octets n'est pas exactement égal à la valeur de « length », le terminal de gestion ne doit pas utiliser le fichier mais doit indiquer à l'utilisateur qu'il est corrompu.

NOTE 4 On ne s'attend pas à ce que le terminal de gestion exécute une quelconque action avec la séquence de valeurs d'octet autre que son téléchargement dans une zone de l'unité gérée.

5.2.5 Format pour les fichiers de produit

Les composants logiciels à télécharger doivent être accompagnés d'un fichier texte, appelé « fichier de produit » contenant des informations concernant un ou plusieurs types de produit.

Le fichier doit contenir une ou plusieurs lignes constituées chacune de zéro ou plusieurs caractères codés selon l'ISO/CEI 646. Les lignes doivent être séparées par un retour chariot (code $0D_{16}$). Un saut de ligne (code $0A_{16}$) peut suivre le retour chariot et doit être ignoré dans ce cas.

La première ligne doit être « `#62379.iec.ch::productfile:formatversion1` ». Les lignes suivantes qui commencent par un « # », ainsi que les lignes vides sont des « commentaires » et doivent être ignorées. (La première ligne ne doit pas être vide.)

Hormis la première ligne et les lignes de commentaires, chaque ligne doit être constituée d'un mot-clé, suivi de manière facultative du caractère « = » et d'une valeur. Lors de l'interprétation d'une ligne, le mot-clé doit être tout ce qui précède le premier « = » et la valeur doit être tout ce qui suit le premier « = »; dans chaque cas, les caractères d'espace blanc qui précèdent et qui suivent (codes 00_{16} à 20_{16} inclus) doivent être supprimés et toute séquence interne de caractères d'espace blanc doit être remplacée par un seul espace. S'il n'y a pas de « = » sur la ligne, le mot-clé doit être la ligne complète (après retrait/remplacement des caractères d'espace blanc) et la valeur doit être vide. Les mots-clés doivent être indépendants de la casse.

Les informations concernant chaque produit doivent commencer par une ligne avec le mot-clé « `product` » et une valeur constituée de quatre nombres hexadécimaux séparés par des espaces, qui sont les quatre nombres constituant `unitIdentity`. Elles doivent inclure tout ce qui précède la ligne suivante comprise avec le mot-clé « `product` », ou jusqu'à la fin du fichier s'il n'y a plus de telles lignes.

Il peut exister une ligne avec le mot-clé « `description` », dont la valeur doit être une description du produit lisible par un être humain.

Pour chaque composant logiciel, il doit y avoir une ligne avec un mot-clé constitué de « `software` », un espace, le type de zone en hexadécimal, un autre espace et la classe de zone en hexadécimal. La valeur doit être « l'emplacement » du fichier contenant le composant de ce type; le format d'un « emplacement » est différent pour les deux méthodes de distribution, voir 5.2.6.

EXEMPLE: La ligne « `software 90 02 = 4-1-2-0.txt` » montre qu'il convient que le logiciel dans la zone de type 90 et la classe 02 (tous deux en hexadécimal) soit comme dans le fichier `4-1-2-0.txt`. Le mot-clé est « `software 90 02` » et la valeur est « `4-1-2-0.txt` ». Il peut y avoir un nombre quelconque de caractères d'espace blanc (ou aucun) de part et d'autre du « = » et à chaque extrémité de la ligne, et toute séquence de caractères d'espace blanc (mais au moins un) de part et d'autre du « `90` ».

NOTE 1 Le fichier lui-même est comme spécifié en 5.2.5 et ne comporte pas de type ou classe, le même composant peut être utilisé à différents emplacements avec des valeurs de type différentes.

NOTE 2 En comparant la partie qui suit le dernier « ~ » dans la première ligne de chaque fichier de composant logiciel avec le `versionNumber` de la zone courante du même type du tableau de « zones », le terminal de gestion peut découvrir si le logiciel correct a été téléchargé dans l'unité gérée. L'interprétation des numéros de version est spécifique au produit, de sorte qu'il convient que le terminal de gestion ne tente pas de vérifier si le logiciel téléchargé est « plus ancien » que le logiciel existant.

5.2.6 Distribution du logiciel

5.2.6.1 Emballé

Le fichier de produit et les fichiers de composants logiciels peuvent être distribués sous forme d'un paquet, par exemple sur un CD ou en envoyant par courrier électronique un fichier ZIP.

Lorsqu'ils sont distribués de cette manière, un « emplacement » doit être un nom de fichier dans le paquet.

NOTE Lorsqu'un nouveau logiciel est disponible, il convient de distribuer un nouveau paquet.

5.2.6.2 En ligne

Le fichier de produit distribué avec le matériel peut être un fichier de produit « indirect » ne spécifiant pas le logiciel direct; à la place des lignes avec les mots-clés « `software` » il doit y avoir une simple ligne avec le mot-clé « `URL` » et la valeur d'un URL pour le fichier de produit « courant ».

Il convient que le fichier de produit « courant » soit également indirect et contienne l'URL d'un fichier de produit pour une version spécifique du logiciel.

Lorsqu'il est distribué de cette manière, un « emplacement » doit être un URL.

Lorsqu'un nouveau logiciel est disponible, il convient de le télécharger sur un serveur et de modifier le fichier de produit « courant » pour pointer sur celui-ci.

NOTE Un terminal de gestion peut vérifier l'existence d'un nouveau logiciel en téléchargeant périodiquement le fichier de produit « courant » et en vérifiant s'il a changé. Il s'ensuit que le fichier de produit « courant » doit être modifié lorsque le logiciel change; il ne suffit pas de télécharger simplement les nouveaux fichiers de composants à la place des anciens.

5.3 Opérations programmées

5.3.1 Demande d'une opération programmée

Une unité gérée doit créer une nouvelle entrée dans sa `scheduledOpTable` lorsqu'une opération `set` est exécutée par une unité de gestion satisfaisant aux conditions suivantes:

- l'un des liens variables se réfère à une instance d'objet ayant un préfixe de type `scheduledOpEntry;`
- la valeur d'index utilisée pour identifier l'instance d'objet susmentionnée ne correspond pas à une entrée existante;
- la valeur d'objet fournie pour l'instance d'objet susmentionnée est une valeur acceptable;
- la réponse à l'opération `set` a un état d'erreur de `noError`.

Toutes les valeurs d'objet de la nouvelle entrée qui ne sont pas fournies par d'autres liens variables dans la même opération `set` doivent être initialisées par l'unité gérée comme décrit ci-dessous.

Si une valeur est fournie pour `soRelativeId`, toute valeur `OperationId` référençant une opération programmée en attente sur l'unité gérée doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser une valeur constituée uniquement de zéros.

Si une valeur est fournie pour `soType`, toute valeur `OperationType` valable prise en charge par l'unité gérée doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser la valeur `modify`.

Si une valeur est fournie pour `soPersistent`, toute valeur `TruthValue` valable doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser la valeur `false`.

Si une valeur est fournie pour `soDateTime`, toute valeur `DateTime` valable ne nécessitant pas l'exécution de l'opération programmée dans les deux minutes suivantes doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser une valeur constituée de l'heure courante plus 5 min pour les opérations absolues ou une valeur de 5 min pour les opérations relatives.

Si une valeur est fournie pour `soObjectName`, toute valeur `ObjectName` se référant à une instance d'objet de la MIB (Base d'informations de gestion, en anglais « Management information base ») mise en œuvre par l'unité gérée et pouvant être écrite (pour des opérations de modification) ou lue (pour des opérations de surveillance) au niveau de privilège spécifié pour `soPrivilege` peut être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser une valeur nulle.

NOTE L'unité peut mettre en œuvre cette disposition d'une manière générale permettant d'écrire tout objet par une opération de modification ou de le lire par une opération de surveillance (soumise à des restrictions de privilège, etc.) ou peut prendre en charge seulement un ensemble spécifique d'opérations programmées. Dans ce dernier cas, seules les valeurs de `soObjectName` qui sont appropriées aux opérations prises en charge seront acceptables.

Si une valeur est fournie pour `soObjectValue`, toute valeur `OCTET STRING` qui, lorsqu'elle est décodée en utilisant les BER (Règles de codage de base, en anglais « Basic encoding rules ») ASN.1, produit une valeur pour l'instance d'objet spécifiée par `soObjectName` doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser une valeur nulle.

Si une valeur est fournie pour `soDescription`, toute valeur `Utf8String` valable doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser une chaîne nulle.

Si une valeur est fournie pour `soPrivilege`, tout niveau de privilège inférieur ou égal au niveau de privilège de l'appel de gestion ayant demandé l'opération `set` doit être considéré comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser le niveau de privilège de l'appel de gestion ayant demandé l'opération `set`.

Si une valeur est fournie pour `soState`, seule la valeur `pending` doit être considérée comme acceptable. Si aucune valeur n'est fournie, l'unité gérée doit utiliser la valeur `pending`.

Une opération `set` tentant d'écrire dans une entrée non existante dans `scheduledOpTable` mais ne satisfaisant pas à toutes les conditions ci-dessus doit être rejetée par l'unité de gestion.

Tandis que la valeur de `soState` dans une entrée dans une `scheduledOpTable` de l'unité est à `pending` ou `delayed`, toute instance d'objet inscriptible associée à cette entrée peut être modifiée par des opérations `set` suivantes, à condition que le niveau de privilège de l'appel de gestion effectuant la modification soit supérieur ou égal à la valeur de `soPrivilege` dans cette entrée et que l'opération programmée associée ne doive pas être exécutée dans les 2 min qui suivent. Les valeurs acceptables pour de telles instances d'objets doivent être comme décrit ci-dessus, à l'exception du fait que toute valeur `OperationState` valable doit être acceptée pour `soState`.

5.3.2 Exécution d'une opération programmée

À chaque fois que la valeur courante de `timeOfDay` est supérieure ou égale à la valeur de `soDateTime` pour une opération absolue donnée et que la valeur de `soState` pour cette opération est `pending`, l'opération doit être exécutée. L'exécution d'une opération de modification doit comprendre le réglage de l'instance d'objet spécifiée par `soObjectName` à la valeur spécifiée par `soObjectValue`. L'exécution d'une opération de surveillance doit comprendre l'attente que la valeur de l'instance d'objet spécifiée par `soObjectName` soit égale à la valeur spécifiée par `soObjectValue`.

Après avoir exécuté une opération, toute opération relative dont la valeur `soRelativeId` fait référence à l'opération exécutée doit être convertie en une opération absolue en réglant sa valeur `soRelativeId` à tout à zéro et en additionnant la valeur courante de `timeOfDay` à sa

valeur `soDateTime`. Si une opération convertie est une opération persistante, les valeurs d'origine de `soRelativeId` et `soDateTime` doivent être mémorisées pour utilisation future.

Pour une opération transitoire, l'entrée de l'opération exécutée doit ensuite être enlevée de la `scheduledOpTable` de l'unité. Pour une opération persistante, les valeurs d'origine de ses `soRelativeId` et `soDateTime` doivent être rétablies (si nécessaire) et s'il s'agissait à l'origine d'une opération absolue, son `soState` doit être mis à `delayed`.

5.3.3 Retard d'une opération programmée

En conséquence d'une opération `set`, lorsque la valeur de `soState` dans une entrée existante d'une `scheduledOpTable` d'une unité est fixée à `delayed`, l'unité doit retarder indéfiniment l'exécution de l'opération programmée associée. `soState` ne peut être modifié que si le niveau de privilège de l'appel de gestion effectuant la modification est supérieur ou égal à la valeur de `soPrivilege` dans cette entrée.

5.3.4 Interruption d'une opération programmée

Lorsque, en conséquence d'une opération `set`, la valeur de `soState` dans une entrée existante de la `scheduledOpTable` d'une unité est fixée à `aborted`, l'entrée de l'opération interrompue doit être enlevée de la `scheduledOpTable` de l'unité.

5.3.5 Etat des opérations relatives

À tout moment où `soRelativeId` d'une opération relative n'est pas égal à `soRequestId` d'une quelconque entrée dans le tableau d'opérations programmées, la valeur de `soState` pour cette opération doit être fixée à `delayed`.

6 Diffusions d'état

6.1 Généralités

Une diffusion d'état doit être initialisée par une unité en réponse à une commande de gestion appropriée ou (si le réseau la prend en charge) appeler une demande de configuration.

NOTE 1 Le processus d'initialisation d'une diffusion d'état et le procédé de transport des informations d'état sont spécifiques au réseau et sont décrits dans la partie secondaire appropriée de la CEI 62379-5 ou de la CEI 62379-6.

Lorsqu'une diffusion d'état a été initialisée, l'unité source doit transmettre une ou plusieurs « pages » d'informations d'état à intervalles réguliers. L'ensemble de pages à transmettre est un « groupe de pages » qui doit être identifié par un OID (Identifiant d'objet, en anglais « Object identifier ») globalement unique. Chaque page du groupe doit être identifiée par un « numéro de page » et la spécification d'un groupe de pages doit définir le format et la sémantique associés à chaque numéro de page.

Les deux premiers octets de chaque page doivent contenir le numéro de page, qui doit être codé par une valeur entière sans signe dans la plage de 1 à 65535 inclus (octet le plus significatif en premier). Les octets restants doivent être codés conformément au format de page. La longueur maximale d'une page doit être de 484 octets.

NOTE 2 Le numéro est le même à chaque fois qu'une page est diffusée, même si elle contient des informations différentes. Par exemple, 6.3.3 spécifie que la page 4 rapporte toutes les opérations correspondantes dans la liste en rotation, de sorte que s'il y a plusieurs de ces opérations, elle en rapporte une différente à chaque fois que la page est transmise; toutefois, dans chaque cas, le numéro de page est codé par 4.

Les entiers doivent être codés en binaire en utilisant un nombre d'octets fixe, le premier octet transmis contenant les 8 bits les plus significatifs de la valeur. Les valeurs n'ont pas de signe,

sauf indication contraire; les nombres avec signe doit être codés en utilisant une représentation par complément à deux.

Les chaînes textuelles doivent être codées en utilisant un codage UTF-8, le premier octet transmis étant le premier octet de la chaîne codée.

Une unité doit théoriquement contenir une source d'informations d'état produisant des pages concernant la totalité de l'unité et pour chaque bloc de l'unité pour lequel des diffusions d'état sont prises en charge, une source d'informations d'état produisant des pages concernant ce bloc. La source d'état doit être identifiée lorsque l'initialisation d'une diffusion d'état est demandée.

Chaque page d'un groupe peut être diffusée à intervalles réguliers ou peut être diffusée à la demande (en étant cependant limitée à une vitesse maximale). La vitesse de diffusion peut être différente pour chaque page d'un groupe. Chaque groupe de pages doit avoir une « vitesse de page de base » associée et la spécification du groupe de pages doit associer la vitesse de diffusion (ou vitesse maximale) pour chaque page de ce groupe à la vitesse de page de base et doit définir une valeur par défaut de la vitesse de page de base.

La vitesse de page de base pour un appel de diffusion d'état particulier peut être spécifiée lors de l'initialisation de l'appel en complétant l'OID identifiant le groupe de pages requis par un simple nombre représentant la vitesse de page de base en pages par minute. Si la vitesse de page de base n'est pas spécifiée, on doit prendre comme hypothèse la vitesse de page de base par défaut. Une unité peut limiter les valeurs de vitesse de page de base qu'elle prend en charge mais doit au minimum prendre en charge la vitesse de page de base par défaut pour chaque groupe de pages qu'elle met en œuvre. L'unité peut diffuser à une vitesse différente si la vitesse de page de base demandée n'est pas prise en charge ou si la page est déjà diffusée à une vitesse différente.

Une unité doit prendre en charge la diffusion simultanée de toutes les combinaisons valables de sources d'état et de groupes de page pour les sources d'état et les groupes de pages qu'elle met en œuvre. Elle doit rejeter toute demande de diffusion d'état avec une combinaison non valable de source d'état et de groupe de pages.

NOTE 3 Il est seulement nécessaire qu'unité prenne en charge une diffusion unique de chaque groupe de pages depuis chaque source d'état. Une demande de diffusion d'état d'un groupe de pages déjà diffusé par la source demandée à une vitesse de page de base différente peut être rejetée si la vitesse de page de base de la diffusion existante n'est pas acceptable par le demandeur.

Si la longueur d'une page reçue est supérieure à ce qui est impliqué par sa définition, le matériel de réception doit ignorer les octets en excès. Si la longueur d'une page reçue est inférieure à ce qui est impliqué par sa définition, le matériel de réception doit remplir les octets manquants avec des zéros.