IEC 62379-1

Edition 1.0 2007-08

INTERNATIONAL STANDARD

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

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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.

- Catalogue of IEC publications: www.iec.ch/searchpub

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 It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

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

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 62379-1

Edition 1.0 2007-08

INTERNATIONAL STANDARD

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

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE XB

ICS 33.160; 35.100

ISBN 2-8318-9279-1

CONTENTS

FOREWORD 4

0 Introduction 6
0.1 Overview 6
0.2 Structure of the family of standards 6
0.3 Model of the equipment being controlled 7
0.3.1 Blocks 7
0.3.2 Control framework 8
0.3.3 How the framework helps when designing units 9
0.3.4 How the framework enables "plug and play" 9
0.3.5 Defining a new type of block 9
0.4 Management information base (MIB) 10
0.4.1 Objects 10
0.4.2 Other uses of OIDs 10
0.4.3 Migration to XML 10
0.5 Status broadcasts 11
0.5.1 Introduction 11
0.5.2 Status page information sources 11
0.5.3 Status page general format 11
0.6 Calls 11
0.7 Privilege levels 12
0.8 Automation 13
0.9 Uploading software 13
0.10 Encapsulation of messages 14
0.11 Further information 14
1 Scope 15
2 Normative references 15
3 Terms and definitions 15
4 Unit management 18
4.1 Protocol 18
4.2 MIB definitions 19
4.2.1 General 19
4.2.2 Application-wide type definitions 20
4.2.3 Conceptual row type definitions 24
4.2.4 MIB objects for basic unit identity and status information 25
4.2.5 MIB objects for the block framework 28
4.2.6 MIB objects for real time clock information 30
4.2.7 MIB objects for reference clock information 31
4.2.8 MIB objects for software upload 32
4.2.9 MIB objects for scheduled operations 34
5 Procedures 36
5.1 Real-time clocks 36
5.2 Procedures for uploading software 36
5.2.1 Areas 36
5.2.2 Contents 37
5.2.3 Procedure for updating a software component 37

5.2.4 File format for a software component 38
5.2.5 Format for product files 39
5.2.6 Software distribution 40
5.3 Scheduled operations 40
5.3.1 Requesting a scheduled operation 40
5.3.2 Executing a scheduled operation 42
5.3.3 Delaying a scheduled operation 42
5.3.4 Aborting a scheduled operation 42
5.3.5 State of relative operations 42
6 Status broadcasts 42
6.1 General 42
6.2 Page formats 44
6.2.1 Basic unit identity page 44
6.2.2 Time-of-day page 44
6.2.3 Scheduled operations page 45
6.3 Page groups 45
6.3.1 `basicUnitStatus` 45
6.3.2 `timeOfDay` 45
6.3.3 `scheduledOps` 45

Annex A (informative) Background information 47
Annex B (informative) Machine-readable MIB definitions 50
Annex C (informative) Machine-readable status page-group definitions 68

Bibliography 69

Figure 1 – A block 7
Figure 2 – Ports 7
Figure 3 – Example of a "unit" 8

Table 1 – Managed objects conveying information about the unit 25
Table 2 – Managed objects for block and connector configuration 28
Table 3 – Managed objects for real-time clock information 30
Table 4 – Managed objects for reference clock information 31
Table 5 – Managed objects for software upload 32
Table 6 – Managed objects for scheduled operations 34

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.

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.

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.

A bilingual version of this publication may be issued at a later date.

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.

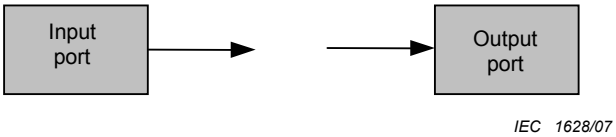

Figure 2 – Ports

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.

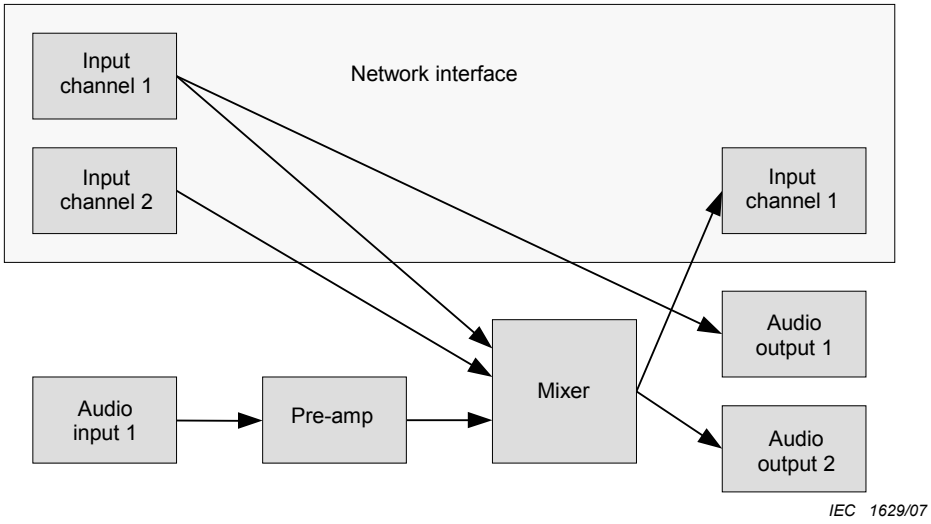

Figure 3 – Example of a "unit"

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)

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
P.O. Box 131
CH-1211 Geneva 20
Switzerland

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