INTERNET-DRAFT
Expires: February 1999

J. Ott/C. Perkins/D. Kutscher
Universitaet Bremen/UCL/Universitaet Bremen

August 1998

A Message Bus for Conferencing Systems

draft-ietf-mmusic-mbus-transport-00.txt

Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited.

Abstract

In a variety of scenarios, a local communication channel is desirable for conference-related information exchange between co-located but otherwise independent application entities, for example those taking part in application sessions that belong to the same conference. Such a mechanism allows for coordination of applications entities to e.g. implement synchronization between media streams or realize *tightly coupled conferences*. The local conference *Message Bus (Mbus)* provides a means to achieve the necessary amount of coordination between co-located conferencing applications for virtually any type of conference. The Message Bus comprises two logically distinct parts: a message transport and addressing infrastructure and a set of common as well as media tool specific messages. This documents deals with message addressing, transport, and security issues and defines the message syntax for the *Mbus*. It does *not* define application oriented semantics and procedures for using the message bus. The common procedures for Mbus operation as well as the common set of application/media specific messages are introduced in a companion Internet draft[9].

This document is intended for discussion in the Multiparty Multimedia Session Control (MMUSIC) working group of the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at confctrl@isi.edu and/or the authors.

1. Introduction

1.1. Background

In the Mbone community a model has arisen whereby a set of loosely coupled tools are used to participate in a conference. A typical scenario is that audio, video and shared workspace functionality is provided by three separate tools (although some combined tools exist). This maps well onto the underlying RTP [5] (as well as other) media streams, which are also transmitted separately. Given such an architecture, it is useful to be able to perform some coordination of the separate media tools. For example, it may be desirable to communicate playout-point information between audio and video tools, in order to implement lip-synchronisation, to arbitrate the use of shared resources (such as input devices), etc.

A refinement of this architecture relies on the presence of a number of *media engines* which perform protocol functions as well as capturing and playout of media. In addition, one (or more) (separate) user interface agents exist that interact with and control their media engine(s). Such an approach allows flexibility in the user-interface design and implementation, but obviously requires some means by which the various involved agents may communicate with one another. This is particularly desirable to enable a coherent response to a user's conference-related actions (such as joining or leaving).

Although current practice in the Mbone community is to work with a loosely coupled conference control model, situations arise where this is not appropriate and a more tightly coupled wide-area conference control protocol must be employed (e. g. for IP telephony). In such cases, it is highly desirable to be able to re-use the existing tools (media engines) available for loosely coupled conferences and integrate them with a system component implementing the tight conference control model. One appropriate means to achieve this integration is a communication channel that allows a dedicated conference control entity to ‘‘remotely’’ control the media engines in addition to or instead of their respective user interfaces.

The Message Bus defined in this and a companion document provides a suitable means for local communication that serves all of the above purposes.

1.2. Purpose

Two components constitute the Message Bus: the (lower level) message passing mechanisms and the (higher level) messages and their semantics.

The purpose of this document is to define the characteristics of the basic Mbus message passing mechanism which is common to all Mbus implementations. This includes the specification of

- the generic Mbus message format;
- the addressing concept for application entities;

- the transport mechanisms to be employed for conveying messages between (co-located) application entities;
- the security concept to prevent misuse of the Message Bus (as taking control of another user's conferencing environment); and
- the details of the Mbus message syntax.

1.3. Terminology for requirement specifications

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant Mbus implementations.

1.4. Definition of terms

- Conference

  The relationship between a set of human beings that are communicating together. In this document, the term is used for both tightly and loosely coupled [4] computer based conferences.
- Participant

  A (typically human) being that takes part in a conference.
- Member

  The system, including all software and hardware components, that is used in a teleconference to represent a single participant.
- End system

  A host or a set of locally interconnected hosts[1] that is used as an interface to a teleconference by a single participant. The end system runs all the required conferencing software (e. g. media agents, session directory, and a controlling entity). End system and software together constitute a member in each of the conferences a user participates in.
- Conference controller

  A dedicated application running on an end system that implements a horizontal conference control protocol through which it interacts with conference controllers on other end systems to implement (typically tight) conference control mechanisms and conference policies. The conference controller constitutes the electronic representation of its (human) user and her actions with respect to conference(s) as a whole (rather than with respect to individual media sessions).

[1] In this document, we use the term "end system" as a synonym for "host" in the simplest case. We do not want to exclude, however, that the local system that serves one participant may be composed of several "hosts" in the Internet sense.

- UCI

  A universal communication identifier of a person. This may be the e-mail address of an individual (or some other globally unique identifier) that is part of the information to identify her within a conference but can also be used to invite her via the Session Initiation Protocol (SIP) [6] protocol.

- Presence

  A presence corresponds to a person (identified by a UCI) being “logged in” at an end system and available for conferencing, i. e. a presence may be identified by the pair of a user’s UCI and the respective end system’s identification (such as a host name). A presence of a user may *appear* in many conferences (see below).

- Appearance

  An instantiation of a user’s presence actually participating (i. e. *appearing*) in a conference is referred to as an *appearance*. There is a one-to-one correspondence between appearances and members.

- Conference context

  All state information kept about a conference at each member of this conference.

- Application session (AS), Session

  The set of media agents/applications that act as peers to each other within a conference. For real-time data, this generally will be an RTP session [5]; for other application protocols, other horizontal protocols may define their own type of session concept. Possible synonyms are “application group” or “media agent group”.

- Application instance, application entity, media agent

  A program instance taking part in an application session for a conference participant. There can be more than one instance of the same program in one session, there can also be more than one instance in different sessions.

2. Requirements and Concepts

The Mbus is supposed to operate in a variety of scenarios as outlined in the introduction. From these scenarios, the following (minimum) requirements are derived that have to be met by the Mbus design to provide a suitable local communication infrastructure.

Local coordination involves a widely varying number of entities: some messages may need to be destined for all local application entities, such as membership information, floor control notifications, dissemination conference state changes, etc. Messages may also be targeted at a certain application class (e.g. all whiteboards or all audio tools) or agent type (e.g. all user interfaces rather than all media engines). Or there may be any (application- or message-specific) subgrouping defining the intended recipients, e.g. messages related to media synchronization. Finally there will be messages that are directed to a single entity, for example, specific configuration settings that a conference controller sends to a application entity or query-response exchanges between any local server and its clients.

The *Mbus* concept as presented here satisfies these different communication models by defining different *message transport* mechanisms (defined in section 3.4) and by providing a flexible *addressing scheme* (defined in section 3.2).

Furthermore, Mbus messages exchanged between application entities may have different reliability requirements (which are typically derived from their semantics). Some messages will have a rather informational character conveying ephemeral state information (which is refreshed/updated periodically), such as the volume meter level of an audio receiver entity to be displayed by its user interface agent. Certain Mbus messages (such as queries for parameters or queries to local servers) may require a response from the peer(s) thereby providing an explicit acknowledgment at the semantic level on top of the Mbus. Other messages will modify the application or conference state and hence it is crucial that they do not get lost. The latter type of message has to be delivered reliably to the recipient, whereas message of the first type do not require reliability mechanisms at the Mbus transport layer. For messages confirmed at the application layer it is up to the discretion of the application whether or not to use a reliable transport underneath.

In some cases, application entities will want to tailor the degree of reliability to their needs, others will want to rely on the underlying transport to ensure delivery of the messages — and this may be different for each Mbus message. The Mbus message passing mechanism described in this paper provides a maximum of flexibility by providing reliable transmission achieved through transport-layer acknowledgments (in case of point-to-point communications only) as well as unreliable message passing (for unicast, local multicast, and local broadcast). We address this topic in section 3.2.

Finally, accidental or malicious disturbance of Mbus communications through messages originated by applications from other users needs to be prevented. Accidental reception of Mbus messages from other users may occur if either two users share the same workstation for conferencing or are using end systems spread across the same physical network: in either case, the Mbus multicast address and the port numbers may match leading to reception of the other party's Mbus messages in addition to a user's own ones. Malicious disturbance may happen because of applications multicasting (e.g. at a global scope) or unicasting Mbus messages (which could contain a "TERMINATE CONFERENCE" command). To eliminate the possibility of receiving bogus Mbus messages, the *Mbus* protocol therefore contains message digests for *authentication*. Furthermore, the Mbus allows for encryption to ensure *privacy* and thus enable using the Mbus for local key distribution and other functions potentially sensitive to eavesdropping. This document defines the framework for configuring *Mbus* applications with regard to security parameters in appendix C (Mbus configuration).

3. Message Bus Specification

3.1. Message Format

A conference coordination message comprises a header and a body. The header is used to indicate how and where a message should be delivered, the body provides information and commands to the destination entity. The following information is included in the header:

- The **MsgDigest** is a Base64-encoded[3] calculated hash value of the entire message (starting from the `ProtocolID` field) as described in appendices A (Algorithms) and C (Mbus configuration).
- A fixed **ProtocolID** field identifies the version of the message bus protocol used. The protocol defined in this document is “mbus/1.0”.
- A sequence number **SeqNum** is contained in each message. The first message sent by a source SHOULD have SeqNum equal to zero, and it SHALL increment by one for each message sent by that source. A single sequence is used for all message from a source, irrespective of the intended recipients and the reliability mode selected. SeqNums are decimal numbers in ASCII representation.
- The **TimeStamp** field is also contained in each message and SHALL contain a decimal number representing the time at message construction in seconds since 00:00:00, UTC, January 1, 1970.
- A **MessageType** field indicates the kind of message being sent. The value “R” indicates that the message is to be transmitted reliably and MUST be acknowledged by the recipient, “U” indicates an unreliable message which MUST NOT be acknowledged.
- The **SrcAddr** field identifies the sender of a message. This MUST be a full address, with no wildcards present. The addressing scheme is described in section 3.2.
- The **DestAddr** field identifies the intended recipient(s) of the message. This field MAY contain wildcards and hence address any number (including zero) of application entities. The addressing scheme is described in section 3.2.
- The **AckList** field comprises a list of SeqNums for which this message is an acknowledgment. See section 3.3 for details.

The header is followed by the message body which contains one or more messages to be delivered to the destination entity. The syntax for a complete message is given in section “syntax”.

3.2. Addressing

Each entity on the message bus SHOULD respond to messages sent to one (or more) addresses. Addresses are quad-tuples written as:

```
(MediaType ModuleType AppName AppInstance)
```

where one or more fields MAY be wildcarded (with '*') in some cases. All fields in an address are case sensitive. The **MediaType** element identifies the type of media processed by an application. Currently defined values are:

audio	An RTP audio stream
video	An RTP video stream
whiteboard	A shared whiteboard
editor	A shared text editor
sap	A session announcement tool, using SAP
sip	A session invitation tool, using SIP
h323	An ITU-T H.323 conference controller
rtsp	An RTSP session controller
control	A local coordination entity

Other values are likely to be defined at a later date.

The **ModuleType** element defines a logical part of an application. The value 'ui' denotes the user-interface of an application, and the value 'engine' defines a media/protocol engine, and 'transcoder' defines a media transcoder. Other values may be defined in future.

The **AppName** element identifies the application being used (e. g.: rat, vic, etc.).

The **AppInstance** element is used to distinguish several instances of the same application. This is a per-instance-unique identifier, which is not necessarily an integer. Many Unix applications will use the process-id (PID) number, although this is not a requirement. Note that if an end system is spread across several hosts, the *AppInstance* MUST NOT be the process-id, unless e. g.. the host name or its IP address are included as well. The companion draft "The Message Bus: Messages and Procedures"[9] defines a bootstrap procedure ensuring that entities can track the abandoning and restarting of application instances as long as unique *AppInstance* values are being used.

The following examples illustrate how to make use of the addresses:

(audio ui rat 124)	The user interface of the rat application with instance-id 124
(workspace ui * *)	The user interfaces of all workspace applications
(audio * * *)	All audio applications
(* * rat *)	All instances of the rat application

3.3. Reliability

While most messages are expected to be sent using unreliable transport, it may be necessary to deliver some messages reliably. Reliability can be selected on a per message basis by means of the MessageType field. Reliable delivery is supported for messages with a single recipient only; i. e., all components of the DestAddr field have to be specified, without the use of wildcards.[2]

[2] Disallowing reliable message delivery for messages sent to multiple destinations is motivated by sim-

Each message is tagged with a message sequence number. If the MessageType is "R", the sender expects an acknowledgment from the recipient within a short period of time. If the acknowledgment is not received within this interval, the sender SHALL retransmit the message (with the same message sequence number), increase the timeout, and restart the timer. Messages SHALL be retransmitted a small number of times before the recipient is considered to have failed. If the message is not delivered successfully, the sending application is notified. In this case, it is up to this application to determine the specific action(s) (if any) to be taken.

Reliable messages are acknowledged by adding their SeqNum to the AckList field of a message sent to the originator of the reliable message. Multiple acknowledgments MAY be sent in a single message. It is possible to either piggy-back the AckList onto another message sent to the same destination, or to send a dedicated acknowledgment message, with no other commands.

The precise procedures are as follows:

Sender:

A sender A of a reliable message M to receiver B SHALL transmit the message via multicast or via unicast, keep a copy of M, initialize a retransmission counter N to '1', and start a retransmission timer T (initialized to T_r). If an acknowledgment is received from B, timer T MUST BE cancelled and the copy of M is discarded. If T expires, the message M SHALL BE retransmitted, the counter N SHALL BE incremented by one, and the timer SHALL BE restarted (set to N*T_r). If N exceeds the retransmission threshold N_r, the transmission is assumed to have failed, further retransmission attempts MUST NOT be undertaken, the copy of M SHALL BE discarded, and the sending application SHALL BE notified.

Receiver:

A receiver B of a reliable message from a sender A SHALL acknowledge receipt of the message within a time period T_c < T_r. This MAY be done by means of a dedicated acknowledgment message or by piggy-backing the acknowledgment on another message addressed only to A.

Receiver optimizing: gathering and piggy-backing ACKs

In a simple implementation, B may choose to immediately send a dedicated acknowledgment message. However, for efficiency, it could add the SeqNum of the received message to a sender-specific list of acknowledgments; if the added SeqNum is the first acknowledgment in the list, B shall start an acknowledgment timer TA (initialized to T_c). When the timer expires, B shall create a dedicated acknowledgment message and send it to A. If B is to transmit another Mbus message addressed only to A, it should piggy-back the acknowledgments onto this message and cancel TA. In either case, B should store a copy of the acknowledgment list as a single entry in the per-sender copy list, keep this entry for a period T_k, and empty the acknowledgment list.

---

plicity of the implementation as well as the protocol. Although ACK implosions are not really an issue and losses are rare, achieving reliability for such messages would require full knowledge of the membership for each subgroup which is deemed too much effort.

In case any of the messages kept in an entry of the copy list is received again from A, the entire acknowledgment list stored in this entry is scheduled for (re-)transmission following the above rules.

Constants:
Suggested values are T_r=100ms, N_r=3, T_c=70ms, T_k=((N_r)*(N_r+1)/2)*T_r.

3.4. Transport

All messages are transmitted as UDP messages with two ways of sending messages being possible:

1) local multicast (host-local or link-local, see Appendix “Mbus configuration”) to a fixed, yet to be assigned link-local address of the administratively scoped multicast space as described in RFC 2365 [8]. There is a base port for each presence conducting conferences using the Mbus. This port SHALL be used for communication between application entities not associated with a particular conference. For each conference that a person participates in, a dedicated port is used for conference-specific communication. Messages of interest for all conferences a presence is involved in SHALL be sent to the base port. Messages intended for a specific conference (i.e. messages relating to an appearance only) SHALL be sent to the port of the respective conference. Message intended for several but not all conferences SHALL be sent individually to the specific ports of these conference (one by one). The concrete port numbers are taken from a reserved set of ports from a defined PORTBASE to PORTBASE+#ports. Appendix B (Port Allocation) defines procedures for port allocation.
2) Directed unicast via UDP to the port of a specific application. This still requires the DestAddr field to be filled in properly. Directed unicast is intended for use in situations where node local multicast is not available. It MAY also be used by Mbus implementations for delivering messages addressed at a single application entity only — the address of which the Mbus implementation has learned from other message exchanges before.

If a single multimedia conferencing endpoint is distributed across several co-located hosts, link local scope SHALL be used for multicasting Mbus messages that potentially have recipients on the other hosts. The Mbus protocol is not intended (and hence deliberately not defined) for communication between hosts not on the same link.

Since messages are transmitted in UDP datagrams, a maximum size of 64 KBytes MUST NOT be exceeded. It is RECOMMENDED that applications using a non host-local scope do not exceed a message size of the network’s MTU.

3.5. Message Syntax

3.5.1. Message Encoding

All messages SHALL use the UTF-8 character encoding. Note that US ASCII is a subset of UTF-8 and requires no additional encoding, and that a message encoded with UTF-8 will not contain zero bytes.

Each Message MAY be encrypted using a secret key algorithm as defined in appendix A (Algorithms).

3.5.2. Message Header

A message starts with the header. The first field in the header is the message digest calculated using a keyed hash algorithm as described in appendix A followed by a newline character. The other fields in the header are separated by white space characters, and followed by a newline. The format of the header is as follows:

```
<MsgDigest>
mbus/1.0 <SeqNum> <TimeStamp> <MessageType> <SrcAddr> <DestAddr> \
        <AckList>
```

The header fields are defined in section 3.1.

3.5.3. Command Syntax

The header is followed by zero, or more, messages to be delivered to the application(s) indicated by the DestAddr field. Each message comprises a command followed by a list of zero, or more, parameters, and is followed by a newline.

```
command ( parameter parameter ... )
```

The command name MUST be a ‘symbol’ as defined in the following table. The parameters MAY be any data type drawn from the following table:

DataType	Syntax	Description
Integer	"-"[0-9]+	
Float	"-"[0-9]+"."[0-9]+	
String	"""..."""	See below for escape characters
List	`(DataType DataType ...)`	
Symbol	`[A-Za-z0-9_-.]+`	A predefined protocol value
Data	`"<"data">"`	Opaque Data

Boolean values are encoded as an integer, with the value of zero representing false, and non-zero representing true (as in the ‘C’ programming language).

String parameters in the payload MUST be enclosed in the double quote (”) character. Within strings, the escape character is the backslash (\), and the following escape sequences are defined:

Opaque data is represented as Base64-encoded [3] character strings surrounded by "<" and ">"

Escape Sequence	Meaning
\\	\
\"	"
\n	<newline>

3.6. Messages

The specific messages applications will send using the Mbus are not defined in this document. Currently a companion document[9] is produced defining classes of messages which are of use in certain application areas. Additional documents are expected to follow.

4. Author's Addresses

Joerg Ott <jo@tzi.org>
Universitaet Bremen, TZI, MZH 5180
Bibliothekstr. 1
D-28359 Bremen
Germany
voice +49 421 201-7028
fax +49 421 218-7000

Colin Perkins <c.perkins@cs.ucl.ac.uk>
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom

Dirk Kutscher <dku@tzi.org>
Universitaet Bremen, TZI, MZH 5160
Bibliothekstr. 1
D-28359 Bremen
Germany
voice +49 421 218-7595
fax +49 421 218-7000

5. References

[1] S. Bradner, “Key words for use in RFCs to Indicate Requirement Levels” RFC 2119, March 1997

[2] H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed-Hashing for Message Authentication”, RFC 2104, February 1997

[3] N. Borenstein, N. Freed “MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies”, RFC 1521, September 1993

[4] Mark Handley, Jon Crowcroft, Carsten Bormann, “The Internet Multimedia Conferencing Architecture,” Internet Draft draft-ietf-mmusic-confarch-00.txt, Work in Progress, February 1996.

[5] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” RFC 1889, January 1996.

[6] Mark Handley, Henning Schulzrinne, Eve Schooler, Jonathan Rosennberg, “SIP: Session Initiation Protocol”, Internet Draft draft-ietf-mmusic-sip-07.txt, Work in Progress, July 16, 1998

[7] M. Handley, V. Jacobson, “SDP: Session Description Protocol”, RFC 2327, April 1998

[8] D. Meyer “Administratively Scoped IP Multicast”, RFC 2365, July 1998

[9] J. Ott, C. Perkins, and D. Kutscher, “The Message Bus: Messages and Procedures”, Internet Draft draft-ietf-mmusic-mbus-semantics-00.txt, Work in Progress, August 1998.

Appendix A: Algorithms

Message Authentication

Either MD5 or SHA-1 SHALL be used for message authentication codes (MACs). An implementation MAY provide SHA-1, whereas MD5 MUST be implemented. To generate keyed hash values the algorithm described in [2] MUST be applied with hash values truncated to 80 bits. The resulting hash values SHALL be Base64 encoded (16 characters). The HMAC algorithm works with both, MD5 and SHA-1.

HMAC values, regardless of the algorithm, MUST therefore always consist of *16* Base64-encoded characters.

Hash keys SHALL have a length of 96 bit, that are 20 Base64-encoded characters.

Encryption

Either DES, 3DES (triple DES) or IDEA SHALL be used for encryption. Encryption MAY be neglected for applications, e. g. in situations where license regulations, export or encryption laws would be offended otherwise. However, the implementation of DES is RECOMMENDED as a baseline. DES implementations MUST use the DES electronic codebook (ECB) mode. Chaining modes are not appropriate due to (possible) unreliable message transport. For algorithms requiring en/decryption data to be padded to certain boundaries ASCII code 32 SHALL be used for padding characters. IDEA uses 128-bit keys (24 Base64-encoded characters). DES SHALL be used with 56-bit keys (12 Base64-encoded characters).

The mandatory subset of algorithms that MUST be provided by implementation is DES and MD5.

See appendix C for a specification of notations for Base64-strings.

Appendix B: Port allocation

The reserved Mbus port numbers are in the range from `PORTBASE` to `PORTBASE+(n*(m+1))` (n=number of base ports, m=reasonable maximum number of conferences per presence). The first n ports are reserved for base ports. The set of conference specific ports starts at offset n and has a cardinality of n*m.

Implementations SHALL use the presence-id (see below) to calculate a valid offset to the set of base port numbers for a person's presence. Offsets to conference specific port numbers SHALL be obtained by using the conference name. The conference name is a SDP session name[7] and MUST be known in advance of port allocation.

Base port number calculation SHALL rely on the following algorithm: All UTF-8 octets of the session name are considered for building a sum of their key codes. The offset to the base port number is the result of the modulo division of the sum by n (number of base ports).

Offsets for per-conference port numbers SHALL be calculated analogously: The key codes of the presence-id's characters are summed up and the the offset is obtained by adding the result of modulo dividing the sum by m (number of conference ports per presence). The actual port number is obtained by adding the result to PORTBASE+(n*(baseport offset+1)).

Example:

PORTBASE	= 2000
nr of base ports n	= 10
nr of conference ports m	= 6
session name	= abc
presence-id	= a@b.org
baseport offset	= (97+98+99) % 10
	= 4
baseport	= PORTBASE + 4
	= 2004
conference port offset	= (97+64+99+46+111+114+103) % 6
	= 4
conference port	= PORTBASE + (6* (baseport offset+1))
	+ conference port offset
	= 2034

Appendix C: Mbus configuration

An implementation MUST be configurable by the following parameters:

Encryption key	The secret key used for message encryption.
Hash key	The hash key used for message authentication.
Presence ID	The UCI of the person participating in a conference.
Scope	The Internet scope to be used for sent messages.

The logical structure of the specified parameters is as follows:[3]

hashkey	::= algo-id expiration key
secretkey	::= algo-id expiration key
presence	::= uci
expiration	::= digits
algo-id	::= "NOENCR" | "DES" | "3DES" | "IDEA" | "HMAC-MD5-80" | "HMAC-SHA1-80"
scope	::= "HOSTLOCAL" | "LINKLOCAL"
key	::= base64string
uci	::= alpha

A Base64-String consists of the characters defined in the Base64 char-set [3] including all eventual padding characters, i. e. the length of Base64-string is always a multiple of 4.

Appendix D: Parameter storage

Two distinct facilities for parameter storage are considered: For Unix-like systems a configuration file SHALL be used and for Windows-95/98/NT systems a set of registry entries is defined.

File based parameter storage:

The file name for a Mbus configuration file is "`.mbus`" in the user's home-directory which MAY be overridden by an environment variable called `MBUS`. Implementations MUST ensure that this file has appropriate file permissions that prevent other users to read or write it. The file MUST exist before a conference is initiated. Its contents SHALL be UTF-8 encoded and SHALL be structured as follows:

```
[MBUS]
HASHKEY=<hashkey>
ENCRYPTIONKEY=<secretkey>
PRESENCE=<presence-id>
SCOPE=<scope-id>
```

A key entry MUST be in this notation:

[3] syntactical definitions follow below

```
"("algo-id"," expiration","base64string")"
```

`algo-id` is one of the character strings specified above and `expiration` is a decimal number representing the date that the key invalidates at, notated in seconds counting from 00:00:00, UTC, January 1, 1970.

The `presence-id` is a universal communication identifier (UCI) for a conference participant. This can be a canonical email address like *"dku@tzi.org"*. In case the same UCI is actually used to represent different presences, e. g. to express different affiliations of a person or to let different person use a single-user end-system concurrently, the presence-id MAY be constituted of a UCI and a presence "modifier" like *"dku@tzi.org#0"*, *"dku@tzi.org#1"* and so on. Presence-ids MUST be in the US-ASCII subset of ISO-10646/UTF-8.

An example Mbus-configuration file:

```
[MBUS]
HASHKEY=(HMAC-MD5-80,946080000,MTIzMTU2MTg5MTEyMQ==)
ENCRYPTIONKEY=(DES,946080000,MTIzMTU2MQ==)
PRESENCE=dku@tzi.org
SCOPE=HOSTLOCAL
```

Registry based parameter storage:

For systems lacking the concept of a user's home-directory as a place for configuration files the suggested database for configuration settings (e.g. the Windows9x-, Windows NT-registry) SHALL be used. The hierarchy for Mbus related registry entries is as follows:[4]

`HKEY_CURRENT_USER\Software\Mbone Applications\Mbus`

The entries in this hierarchy section are

Name	Type
HASHKEY	String
ENCRYPTIONKEY	String
PRESENCE	String
SCOPE	String

The same syntax for key values as for the file based configuration facility MUST be used.

[4] complies with vat's registry hierarchy