HOMEMEETING

TECHNICAL GUIDE

HomeMeeting Integration Tools Version 3.23.9

January 2017

A HomeMeeting Inc. Official Document

Disclaimer; No Warranty

THIS INFORMATION AND ALL OTHER DOCUMENTATION (IN PRINTED OR ELECTRONIC FORM) ARE PROVIDED FOR REFERENCE PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THIS INFORMATION, THIS INFORMATION AND ALL OTHER DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT ANY WARRANTY WHATSOEVER AND TO THE MAXIMUM EXTENT PERMITTED, HOMEMEETING INC. DISCLAIMS ALL WARRANTIES, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SAME. HOMEMEETING INC. SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES, ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS INFORMATION OR ANY OTHER DOCUMENTATION. NOTWITHSTANDING ANYTHING TO THE CONTRARY, NOTHING CONTAINED IN THIS INFORMATION OR ANY OTHER DOCUMENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM HOMEMEETING INC. (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF THIS SOFTWARE.

Copyright

CONTENT

1. Background

HomeMeeting Integration Tools (hereafter called HIT) is designed to support a flexible integration between a third-party application and the HomeMeeting server.

As far as a third-party application is concerned, the HomeMeeting server (hereafter called MCU) is a pure meeting service provider and does not directly authenticate any user, instead it receives meeting request from a trusted third-party application. The third-party application could be a web server or any interface interacting with end-users. The third-party application is responsible to maintain user database, authenticate user, construct meeting service request to MCU server, and to provide service interface to its end users.

Once authenticated by the third party, users can request specific services such as meeting, play back or share recording files. For example, when a user specifies a meeting request, the third party then constructs a *fresh* jnj file, allowing the user using this jnj file to connect to the MCU. The user's information and service request is contained in this jnj file. The MCU checks the freshness of this jnj file and the jnj file is processed only if the jnj file is fresh (By default, a jnj file is invalid after *5 minutes* past its generation). It is important to note that this freshness checking requires that the MCU and the third party synchronize their clocks. The mechanism to synchronize clock is out of the scope of this document.

The following is an example of a jnj file generated by a third party application:

```
[general]
domain = HomeMeeting
codetype = 13
ip = 192.168.1.63
action = 0
UserInfo = key_web_192.168.1.63_0000|Bq6=|NOyO1+r1iO7R|nrSn9048=
```

This type of jnj file is vulnerable to passive attack. If an attacker can intercept the jnj file that is transmitted through the network, the malicious attacker can impersonate the real user. To defeat such a passive attack, it is recommended to use a secure transmission such as https to disseminate the jnj file. Note that the 'action' entry in the jnj file shows that the requested service is a meeting (action = 0) or a playback (action = 1); entry 'codetype' is 13, which is a predetermined value for all HIT service requests.

HIT also supports direct authentication between an end-user and MCU. When the direct authentication is used, freshness of the jnj file is not required and so it does not rely on the clock synchronization between MCU server and the third party. Direct authentication can also defeat passive attack because the password input from the user is transmitted through the secure JoinNet-MCU channel. Password authentication is one example of direct authentication.

2. Authentication Relay

HIT uses public key infrastructure to relay the authentication from the third-party application to the MCU.

A specific Diffie-Hellman key exchange algorithm is used to relay the authentication and the service request. For the DH key exchange algorithm, fixed parameters are:

- Prime (in hex):
  FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC740
  20BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374F
  E1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE3
  86BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA
  48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED52
  9077096966D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E77
  2C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497
  CEA956AE515D2261898FA051015728E5A8AACAA68FFFFFFFFFFFFFFFF (This is a 2048 bits prime. For MCU 3.23.0 or lower versions, a 1024 bits prime was used.)
- Generator: 2

Both the MCU and the third party keep a DH key pair, which are used to encrypt and verify the service request payload in the jnj file. The third party first constructs an XML file, which contains the user's information and service request; then this XML file is processed as in Fig. 1.

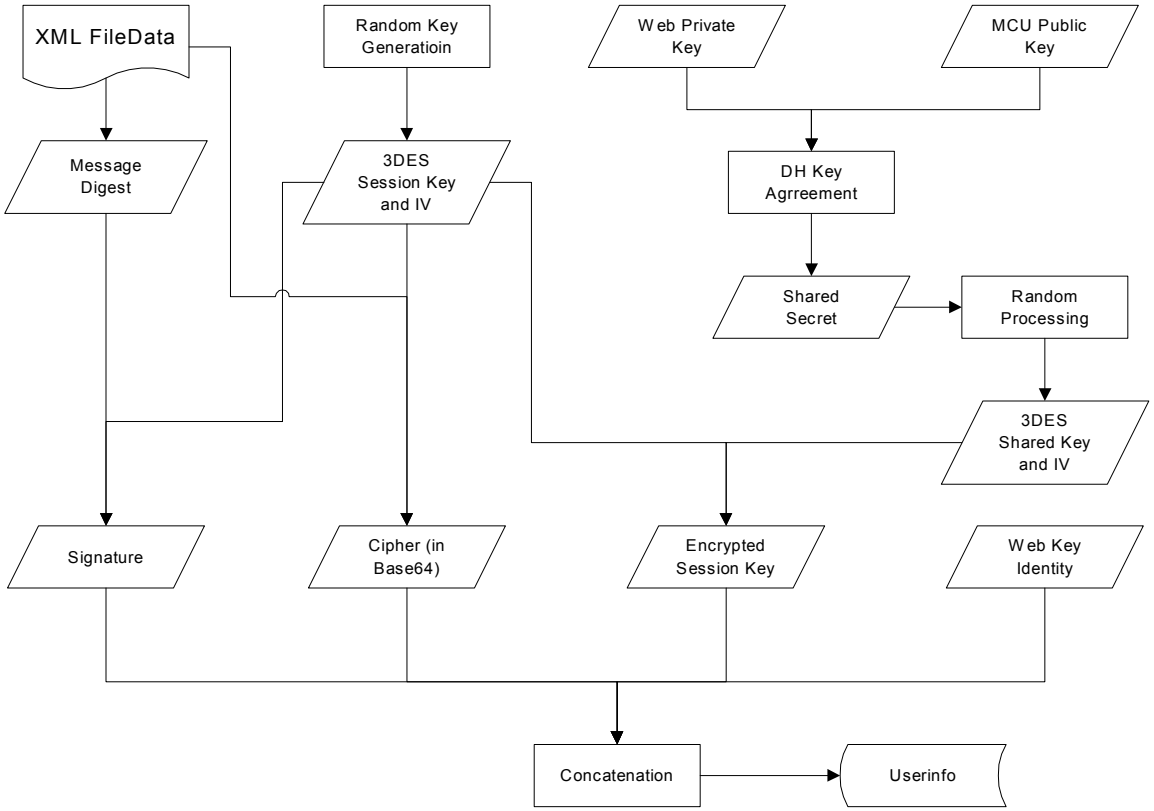

Figure 1. Userinfo Generation

The userinfo is put into the jnj file as one entry. When an end-user opens this jnj file, userinfo is sent to MCU. MCU uses the Web key identity contained in the userinfo to locate the web public key file. If the key file cannot be found, the service request is rejected. MCU decrypt and verify the XML file data contained in the userinfo as in Fig. 2.

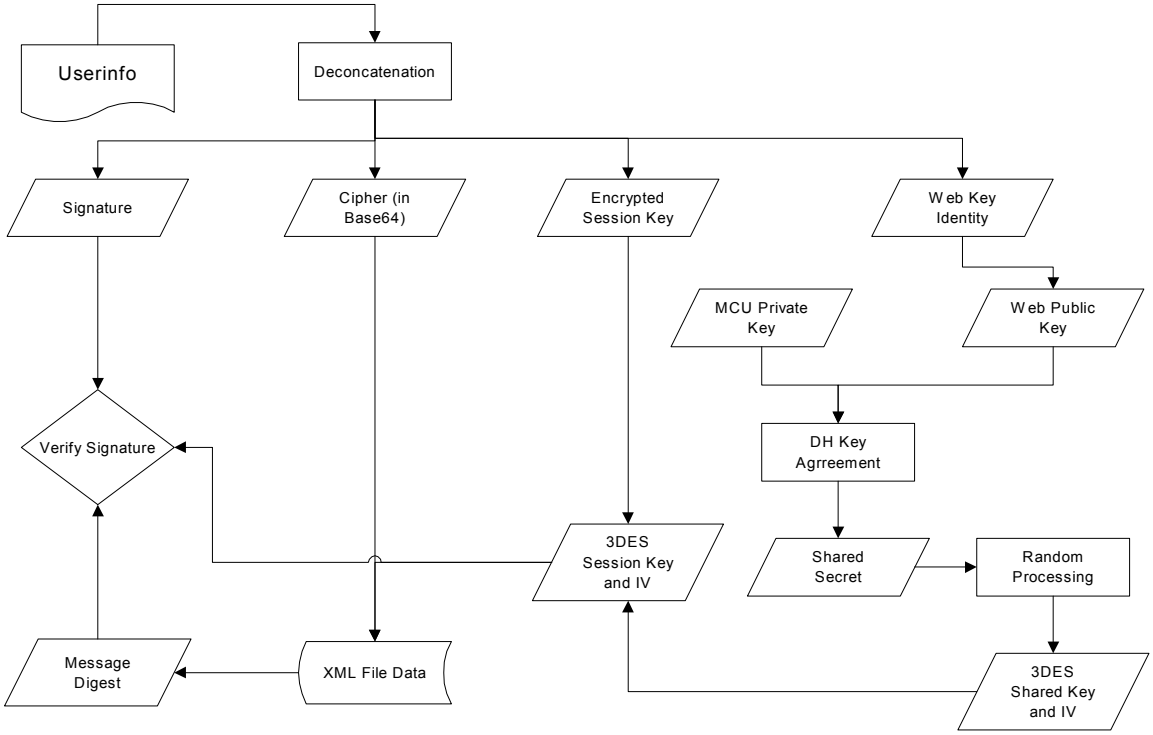

Figure 2. MCU decrypt and verify the userinfo

3. Interface Between The Third Party and MCU

3.1 Service Request in XML Format

The end-user's service request is saved in XML format.

```
<? xml version="1.0" encoding="utf-8" ?>
<jnj>
   <owner id=″ownerid″ diskquota=″1000″ maxoutconnection=″50″
email=″owner@email.com″>ownername</owner>
   <meetingid>testing</meetingid>
   <timestamp>1019077830</timestamp>

   <password>Base64 encoded SHA hash of password</password>
   <guest id="guestid" email=″guest@email.com″ invited="1"
ticket=″0″>guestname</guest>
   <group diskquota="1000" maxoutconnection="50">HomeMeeting</group>
   <command recording="1" exclusive_recording=″0″ autoextension=″0″
duration="60" default_joint_browsing_page=″″ all_questioner=″0″
preparationmode=″1″ bandwidth=″64000″ total_av_bandwidth=″256000″
max_message_duration=″10″ joinnet_auto_update_url=″″
joinnet_holder_video=″1″ audio_codec_preference=″″
video_codec_preference=″″ file="recording_file_name">meeting
      <guaranteed invited=″10″ uninvited=″3″>1</guaranteed>
      <transferslide userid=″userid″ groupid=″groupid″
file=″recording_file_name″>meetingid</transferslide>
      <continuemeeting userid=″userid″ groupid=″groupid″
file=″recording_file_name″>meetingid</continuemeeting>
   </command>
   <meetingtitle>testing title</meetingtitle>
</jnj>
```

Figure 3 Service request example

Fig. 3 shows an example of a service request. There are four required items in the xml format data: "*owner*", "*meetingid*", "*timestamp*" and owner's "*id*". All other elements and attributes are optional.

All the string elements should be coded in utf-8 and the length restrictions are as followed: userid, username, groupid, meetingid, and email should be less than 128 bytes; meeting title, filename, and default joint browsing page should be less than 512 bytes; password should be less than 40 bytes.

When the service request is for meeting purpose, only the first request that actually creates the meeting will set the meeting's corresponding properties. All the following request that join the same meeting will not affect the meeting's existing properties. There are a few exceptions for this rule. The exceptions are listed below.

- The following elements are not saved as meeting's properties: <owner maxoutconnection>, <password>, <timestamp>, <group maxoutconnection>.

- When the service requester is a guest, the exceptions include: <guest>.

- When the service requester is an owner, the exceptions include: <owner>, <owner email>.

- When the service requester is an owner who switches an ongoing preparation mode meeting into a normal meeting, the exceptions include: <owner>, <owner email>, <group>, <meetingtitle>, <command autoextension>, <command duration>, <command guaranteed> <command guaranteed invited>, <command guaranteed uninvited>.

Each element's usage and specification is described in the followings.

1. Element "owner" [*required*]: "owner" specifies the user name of an owner. This name will be shown in the JoinNet's Control Panel.

    - Attribute "id" [*required*]: "id" specifies the owner's userid. The owner's userid should be a unique attribute in the third-party application. If the service request is a meeting request, the disk usage of the recording file is charged to this user and all the participants' network connections are counted to this userid during the meeting.

    - Attribute "diskquota" [optional, default = 50, configurable in configm.ini]: "diskquota" specifies the owner's diskquota (in Mbytes) for recording file storage. When the service request is "meeting" and the meeting is to be recorded, this owner's recording file usage is checked against this "diskquota". The meeting request is rejected when the disk usage exceeds the "diskquota".

      The diskquota check can only be done before the meeting begins. After a meeting finishes and the new recording file is created, the disk usage including this new recording file may exceed the diskquota. However the new recording file is always saved even if this new recording file exceeds the diskquota. This also means that the user's disk usage may *exceed* his/her diskquota. The default value of diskquota is configured in MCU's configm.ini (default_disk_quota). If default_disk_quota is not set in configm.ini, 50 Mbytes are used as the default value.

      The third party can set the special value "0" to "diskquota", which tells MCU not to enforce the diskquota checking. In this case, the third party is in charge of the diskquota enforcement.

There is another configuration in MCU's configuration – "grace disk quota". This configuration is in percent. For example, 20 means 20% grace disk quota. When this configuration is active, the grace disk quota will be taken into consideration when the disk usage exceeds the "diskquota".

- Attribute "maxoutconnection" [optional, default=10, configurable in configm.ini]: "maxoutconnection" specifies the maximal outside connection that is charged to the owner. An outside connection is a connection to the MCU from an IP that is out of the IP range in the license of the MCU. If there is no IP range in the license, all connections are outside connections. The connection that is charged to the owner includes: (a) this owner's connection (both meeting and playback), (b) the connections of other participants in the owner's meeting, and (c) the SIP phone called by any participants in the owner's meeting. When the current outside connection that is charged to the owner exceeds the "maxoutconnection", the service request is rejected.

  The default value of maxoutconnection is configured in MCU's configm.ini (default_max_outconnection) if this attribute is not set. If default_max_outconnection is not set in configm.ini either, 10 are used as the default value.

- Attribute "email" [optional]: "email" specifies the email address of the owner. When the MCU is configured to support email notification, the MCU will send to this "email" address when (a) the owner's disk usage exceeds his/her diskquota. (This is done at the end of meeting and grace disk quota is **NOT** considered in this decision loop), and (b) a guest leaves a message to this owner.

  Note, the email sent by MCU is very preliminary, it is recommended that the third party send these kinds of alert emails to the owner if possible. The MCU will notify the third party through the Messenger interface when these events occur. The Messenger interface is described in the following chapters.

2. Element "meetingid" [*required*]: "meetingid" is used in two scenarios. One is to save the recording file in subdirectory of the "meetingid", so that the third party can locate the recording file by using the "meetingid". The other case is to use "meetingid" to identify a unique meeting.

   "meetingid" is used to identify a specific meeting so that it needs to be unique throughout one owner's lifetime. The third party needs to ensure that the "meetingid" in each jnj file created for a specific user is unique. The "meetingid" can have 128 bytes (not including the ending null), so it is not very difficult to achieve this. Simple implementation can be: (a) use an integer which increase by

one for every created jnj file for meeting and wrap back when reaching the MAX_INT, or (b) use GUID generated by the system—this will cause very long directory name in the recording directory/file structure.

Note that "meetingid" is not used in service request “checkmessage”.

3. Element “timestamp” [*required*]: “timestamp” is used to check the freshness of the service request. If the time contained in “timestamp” is within 5 minutes of MCU’s local time, the service request is considered fresh; otherwise the request is rejected.

   When “password” is present in the xml format data, “timestamp” indicates when the “password” expires (see also Section 3.10).

4. Element “password” [optional]: “password” contains the base64 encoded SHA hash of password. MCU will use this information to authentication end-users. For example, if the password is “testpassword”, the element “password” has value “i7YRj4/Wk1rQh2o740pxfTJwj/0=”.

   When this element is present, element “timestamp” indicates when the “password” expires.

5. Element “guest” [optional]: “guest” specifies the username of a guest. If this element is present in the XML format data, the service requester is considered as a **guest**; otherwise the requester is considered as an **owner**.

   Generally, a guest cannot open a meeting room. Only an owner can start an instant meeting, namely open a person-based meeting room. However, an *invited* guest can open an event-based meeting room when the owner is late to join the meeting. The difference between event-based meeting room and person-based meeting room will be discussed later. When a guest chooses to leave a message to an owner while the owner is not online, the guest actually open a person-based meeting room in the name of the owner for message recording.

   Note, when the command is ‘playback’ or ‘download’, guest “id” and “name” information represent the information of the player or the user downloading the file.

   - Attribute “id” [optional]: “id” specifies the userid of a guest. A guest with “id” in the XML format data is considered as an identifiable guest; otherwise, the guest is anonymous because the username cannot be used to identify a user. Because the “id” is used to identify the guest, it should also be unique as the owner’s “id”.

     In the same meeting, if two guests have the same “id” but different username, they are considered as the ***same*** person, and only one of them

can join the meeting. (To avoid confusion, the third party should prevent this.) On the other hand, in the same meeting, if two guests have no "id" but have same username, they are considered as ***different*** persons and both of them can join the meeting.

Only a guest with "id" can be *permanently* disconnected by the meeting owner.

- Attribute "email" [optional]: "email" specifies the email address of the guest. Only a guest with "id" can be specified with the "email" information.

- Attribute "invited" [optional, default = "0"]: "invited" specifies whether this guest is invited or not. If a guest is invited ("invited" equal to "1"), the guest can join the owner's office without asking the owner's permission. Otherwise, an uninvited guest has to ask the owner's permission to join the owner's meeting. If an owner sets "do not disturb" property in his/her JoinNet, no uninvited guest can enter the owner's office any more.

  Note the attribute "id" and "invited" are independent. So there are four kinds of guest: (a) identifiable and invited, (b) identifiable and uninvited, (c) anonymous and uninvited, and (d) anonymous and invited—this setting is not recommended.

- Attribute "ticket" [optional, default = "0"]: "ticket" specifies whether this guest is a ticket holder. The MCU needs a license supporting ticket-base meeting to accept the service request of a ticket holder. "Ticket" is only used in an event-based meeting. When an owner has not started the meeting, a ticket holder cannot join the event regardless of the value of attribute "invited". When an owner already creates a ticket-base meeting, a ticket holder can join the event without the owner's permission similar as an invited guest; moreover, the ticket holder's connection is counted as an invited guest.

  A ticket holder **cannot** send out video during the meeting. The user information of a ticket holder is only shown in the coordinator's control panel, i.e., the other participant does not see the ticket holder in the user list. The JoinNet feature of auto questioning in a ticket-base meeting is disabled.

  Note, a ticket holder must have the attribute "id", i.e., a ticket holder must be identifiable.

6. Element "group" [optional]: "group" specifies the groupid of the ***owner***. When this element is present in the XML format data, group-related restriction is also

applied to the service request and the recording file is written into the subdirectory of the corresponding "group".

- Attribute "diskquota" [optional, default = 50, configurable in configm.ini]: "diskquota" specifies the group's diskquota (in Mbytes) for recording file purpose. When the service request is "meeting" and the meeting is to be recorded, this group's recording file usage is checked against the "diskquota". The meeting request is rejected when the disk usage exceeds the "diskquota". See detailed discussion in the element "owner".

- Attribute "maxoutconnection" [optional, default=10]: "maxoutconnection" specifies the maximal outside connection. See detailed discussion in the element "owner".

  When the current outside connection that is charged to the group exceeds the "maxoutconnection", the service request is rejected.

  The default value of maxoutconnection is configured in MCU's configm.ini (default_max_group_outconnection) if this attribute is not set. If default_max_group_outconnection is not set in configm.ini either, 10 are used as the default value.

7. Element "command" [optional, default="meeting"]: "command" specifies the requested service. There are seven types of services: (a) meeting, (b) playback, (c) download, (d) delete, (e) checkmessage, (f) talktovisitor, and (g) leavemessage.

   Note that the recording file(s) may be deleted when the service request is 'delete' or 'checkmessage'. If the third-party application has its own recording file management scheme and does not want any end-user to directly delete recording files, the third party should not issue jnj for these two types of service request.

   Command 'talktovisitor' is a special command, which enables an owner (identified by the ownerid) to talk to a visitor who is leaving a message (identified by the meetingid) for the owner. There are several restrictions for issuing this command: the owner should not have an ongoing meeting, and there should be an ongoing meeting with meetingid matching the one specified in the userinfo and owner id matching the owner id specified in the userinfo, respectively. All the attributes within 'command' section are ignored for command 'talktovisitor'.

   Command 'leavemessage' (only issued to visitors) is same as 'meeting' except that the visitor will leave a message to the owner no matter what the owner's office is open or not.

   - Attribute "recording" [optional, default="1"]: "recording" specifies whether a meeting is to be recorded. This attribute is ignored if the

command is not “meeting”. When the service requester is a guest and this attribute is set to “0”, the requester cannot leave message if the owner is not online.

- Attribute “autoextension” [optional, default=”1”]: “autoextension” specifies whether an event-based meeting will be automatically extended after the meeting running time has exceeded the “duration”. If this attribute is set to “0”, the MCU will terminate the meeting after the meeting running time exceeds the “duration”. If this attribute is set to “1”, the meeting will keep running after the running time has exceeded the “duration” as long as there is any active participant in the meeting room. This attribute is ignored if the command is not “meeting” or the attribute “duration” is not present.

- Attribute “duration” [optional]: “duration” specifies the duration in minutes of a meeting. “duration” is the meeting duration in minutes, which should be less than 600 (10 hour). Any duration larger or equal to 600 will be cut to 599. This attribute is ignored if the command is not “meeting”. When this attribute is present and is not “0”, the meeting is considered as an event-based meeting; otherwise, the meeting is considered as a person-based meeting. Table 1 states the differences of person-based and event-based meeting:

Scenario	Person-based meeting	Event-based meeting
An invited guest request meeting service when the owner has not opened the meeting	The guest can leave a message	The guest opens the meeting and joins the meeting. (In this case the guest gets the token)
An uninvited guest request meeting service when the owner has not opened the meeting	The guest can leave a message	The guest get error message "meeting is not started or already over"
When will the meeting be terminated?	1. When the owner exits JoinNet and terminate the meeting explicitly. 2. When all the participants left the room and the room is empty for one minute.	1. When the owner exits JoinNet and terminate the meeting explicitly. 2. When autoextension is set to "0", the meeting running time exceeds the "duration". 3. When autoextension is set to "1", all the participants left the room and the room is empty for about one minute after the meeting running time exceeds the "duration".
All participants leave the meeting so that the meeting room is empty	The MCU server will terminate the meeting. So if any participant comes back, he/she will enter a new meeting room.	The meeting will keep running until the meeting running time exceeds the "duration". So if any participant comes back, he/she will enter the old meeting room.
If two owners use the same meetingid. (Note: this is an error case, which should never happen)	These two owners open each's web office and the meetings will be recorded in separate directories.	The first owner open the meeting room, while the second owner get an error message "Internal error".
If the same owner requests two meeting service using different meetingid. (This may occur when the requests are initiated from different computers)	The first requester creates a web office for the owner, and the second requester enters the same office and kicks out the first requester.	Two different meeting rooms are created.

Table 1. The difference between person-based and event-based meeting

- Attribute "default_joint_browsing_page" [optional]:
  "default_joint_browsing_page" specifies the default joint browsing page for the meeting. This attribute is ignored if the service request is not 'meeting'.

- Attribute "exclusive_recording" [optional, default="0"]: "exclusive_recording" specifies whether the meeting can be launched when there are already old recording files in the recording folder. This attribute is ignored if the command is not "meeting". When the attribute is set and there are already old recording files in the recording folder, the meeting cannot be launched.

- Attribute "all_questioner" [optional]: "all_questioner" specifies a specific type of meeting, i.e., all participants will be questioners all the time. The number of questioners a meeting can support is controlled by the HomeMeeting license and the configm.ini of the MCU. If the current

number of participants has reached this threshold, no more participants will be allowed to join the meeting regardless of the other settings or reservations.

- Attribute “preparationmode” [optional]: “preparationmode” specifies the preparation mode for an event-based meeting. It is ignored for a person-based meeting (i.e. a meeting whose duration is 0). The detail of the usage of this tag is in section “Preparation Mode.”

- Attribute “bandwidth” [optional]: “bandwidth” specifies the maximal bandwidth that the user can use as the video sending rate. The unit is bps.

- Attribute “total_av_bandwidth” [optional]: “total_av_bandwidth” specifies the maximal total bandwidth for audio and video that a meeting can be allocated. When receiving this information, each individual JoinNet should adjust its video sending rate based on the number of participants speaking at the same time. The unit is bps. Note that only JoinNet with jnkernel 1.23 or higher can receive this information.

- Attribute “max_message_duration” [optional]: “max_message_duration” specifies the max duration in minutes that a guest can leave a message. At the 5 minutes mark, the guest will receive a warning from the MCU server that the meeting is going to be terminated soon.

- Attribute “joinnet_auto_update_url” [optional]: “joinnet_auto_update_url” tell the JoinNet where to check auto update. It is a URL with maximal length of 512 bytes (including the ending null). Note that only JoinNet with jnkernel 1.23 or higher can receive this information.

- Attribute “joinnet_holder_video” [optional]: “joinnet_holder_video” set to 1, the JoinNet show token holder’s video on the main window by default. Otherwise, the questioner’s video is shown by default. Note that only JoinNet with jnkernel 1.23 or higher can receive this information.

- Attribute “audio_codec_preference” [optional]: “audio_codec_preference” tells JoinNet which is the audio codec preference for this meeting. The order of the codec list shows the server's preference, i.e. the client should choose the first supported codec by default. The client can choose any supported codec at the user's decision though. To date, the supported audio codec are: dflt, ilbc, g711, g726, opus. The default bitrate(without overhead) of these codec are: dflt(6.4kbps), ilbc(13.5kbps), g711(64kbps), g726(16kbps), opus(configurable). Note that only JoinNet with jnkernel 1.23 or higher can receive this information.

- Attribute “video_codec_preference” [optional]: “video_codec_preference” tells JoinNet which is the video codec preference for this meeting. The

order of the codec list shows the server's preference, i.e. the client should choose the first supported codec by default. The client can choose any supported codec at the user's decision though. To date, the supported video codec are: dflt, hm264, f264, mjpg, vpx. Note only JoinNet with jnkernel 1.23 or higher can receive this information.

- Attribute “file” [optional]: “file” specifies the recording file name (*excluding* the extension “.jnr”). This attribute is only meaningful for command “playback”, “download” and “delete”. Normally, the “meetingid” is used to identify a recording file to playback, download or delete. However, there could be multiple recording files for one single “meetingid”. This attribute is used to uniquely specify the recording file.

  If this attribute is not present and there are multiple recording files for a single “meetingid”, the recording file with the *newest creation date* is chosen.

- Subelement “transferslide” [optional]: “transferslide” specifies the meetingid for a recording file whose slides will be transferred to the meeting. Only a “meeting” command uses this subelement.

  “transferslide” has 3 attributes: “userid”, “groupid” and “file”. “userid”, “groupid” and “meetingid” together can define a recording file, i.e. the recording file for “meetingid” of “userid” and “groupid”. “groupid” is optional if the user does not belong to any group. If there are multiple recording files for a given set of [“userid”,”groupid”,”meetingid”], the newest one is used for transferring slides. The optional attribute “file” can be used to uniquely define a recording file.

- Subelement “guaranteed” [optional]: “guaranteed” specifies whether an event-based meeting uses the guaranteed resource. If this subelement has the value “1”, the two attributes “invited” and “uninvited” define how many invited and uninvited guests are reserved in the meeting. This subelement is ignored if the meeting is not an event-based meeting.

- Subelement “continuemeeting” [optional]: “continuemeeting” specifies the meetingid for a recording file that the meeting is based on. Only a “meeting” command uses this subelement. When this subelement is specified, the recorded information in the recording file will be reloaded to the new-launched meeting and the meeting continues from the end of the recorded data.

  “continuemeeting” has 3 attributes: “userid”, “groupid” and “file”. “userid”, “groupid” and “meetingid” together can determine a recording file, i.e. the recording file for “meetingid” of “userid” and “groupid”. “groupid” is optional if the user does not belong to any group. If there are

multiple recording files for a given set of ["userid","groupid","meetingid"], the newest one is used. The optional attribute "file" can be used to uniquely specify a recording file.

8. Element "meetingtitle" [optional]: This element is only used to write into the corresponding status XML file for each recording file. It is not used for the meeting purpose. The usage of this attribute is discussed later.

3.2 MCU's Configuration

The MCU should be configured correctly to accept the service request generated by a third-party application.

The DH key pair used in the authentication relay can be created by a separate program. After a specific key pair is created, copy the key pair to the MCU directory (the directory that the MCU executable file locates) and set the following entries in MCU's configuration file:

mcu_cluster_dh_key=mcu_key_identity
passphrase=passphrase_of_mcu_priv_key

The passphrase is in the file "passphrase.txt", which is generated when running the key pair generation application.

To accept the service request from a certain third party, just put the third party's public key file in the MCU directory. One MCU can accept service request from multiple third parties by putting multiple public key files in MCU directory.

To invalidate a third-party application, its public key needs to be deleted from MCU directory.

3.3 Recording File Directory Hierarchy

The recording files in the THIRDPARTY scheme is saved in hierarchy. Suppose that the configured recording root directory is "~root". If there is no "group" in the userinfo (XML format data), the recording file is saved as:

~root/_user/ownerid/meetingid /_recording_ip_date_sessionindex.jnr

If there is "group" in the userinfo (XML format data), the recording file is saved as:

~root/_group/groupid/ownerid/meetingid /_recording_ip_date_sessionindex.jnr

The disk usage of a certain group "groupid" is defined as the total file size of the recording files (including all the subdirectories) under directory

~root/_group/groupid

The disk usage of a certain owner "ownerid" is defined as the total file size of the recording files (including all the subdirectories) under directory

~root/_user/ownerid and ~root/_group/*/ownerid

In the above definition, the same "ownerid" under different "groupid" are considered as the same user. So the "ownerid" should be unique throughout the whole system.

3.4 Owner Meeting Status File

The MCU maintains two sets of owner meeting status XML files in the recording directory. The first set of status files adopt the name in the format of:

_status_ip_xxx.xml

The 'xxx' is the local IP of the MCU server. The local IP of a MCU server can be determined by the following steps:

- If the license of the server specify a fixed IP, it is assigned to Local IP;
- Else if "listenip" is present in configm.ini, its value is assigned to Local IP;
- Else if the server has at least one network adapter, the IP of the first network adapter is assigned to Local IP. (i.e. the first entry in the list return by gethostname);
- Else Local IP is assigned "127.0.0.1".

Given a MCU server and its configuration, the Local IP of this server is determinate. This is important because the third party need to know this information to locate the status file of a specific MCU server.

When IPv6 IP address is used, all the ":" in the IPv6 IP address are replaced with ";" when being used in filename. For example, if the MCU's IP address is fe80::215:c5ff:fe0e:bef8, the corresponding file name will be _status_ip_fe80;;215;c5ff;fe0e;bef8.xml. This strategy applies to all the other file names using IP address.

When the "jnj_ip" entry in the configm.ini is set and is different with the Local IP as determined in the above steps, a second status file is also written using the configured "jnj_ip".

There could be at most three status files maintained by a MCU:

_status_ip_listenipv4.xml
_status_ip_listenipv6.xml
_status_ip_jnj_ip.xml

The content of these files are identical.

Here is an example of a status XML file:

```
<?xml version="1.0" encoding="utf-8" ?>
<ThirdParty utc="1477579954" last_update="Fri Apr 26 19:55:28">
   <info max_meeting=”10” max_connection=”20”
max_outside_connection=”10” max_reserved_connection=”5”
max_reserved_outside_connection=”3” ip_range=”209.101.242.123/16”
target_bitrate=”24.0Kbps” test_wizard_link=”wizard_mjnj”/>
   <directory>
      <meeting vid="0" session_index="0" name="John" userid="123"
meetingid="456" duration="30" elapsed_second1="303"
elapsed_second2="303" max_participant="6" owner_present="1"
have_guest="1" preparation_mode="1"/>
      <playback vid="0 session_index="2" name="player"
playerid="123" ip="192.168.1.10" elapsed_second="507"
shared_playback="1" end_reached="1" file_name="C:/Homemeeting/MMC
Server/data/recording/_user/2/169/_recording_192.168.1.21_2016_10_25
_21_07_001835.jnr" file_size="708125" file_duration="1298"
file_encrypted="0"/>
   </directory>
</ThirdParty>
```

The attribute “last_update” states the time that this file is last updated by a MCU server. The attribute "tick" is the same as "last_update", but in machine-friendly format. The attributes of element “info” specify the properties of this MCU server. The third party can use these information to tell end users the server’ configuration. When a MCU server is down, there is no attribute for the element “info”.

The sub-element “meeting”(or "user" when the owner is in meeting) under element “directory” shows the information of the ongoing meeting sessions. The third party can use these information to show owner’s meeting status to end users. The sub-element “playback” under element “directory” shows the information of the ongoing playback sessions.

The meanings of each entry for "meeting" are:

entry	meaning	default value
vid	virtual server id	0
session_index	meeting session index	-1
name	owner's name	empty
userid	owner's internal id	empty
meeting_id	meeting id	empty
duration	scheduled meeting's duration in minutes	0
elapsed_second1	actual logical time passed since the start of meeting, in seconds. The logical time span covers the duration of the old JNR if the meeting is continued from a JNR.	0
elapsed_second2	actual physical time passed since the start of meeting, in seconds. The physical time span doesn't cover the duration of the old JNR if the meeting is continued from a JNR.	0
max_participant	the number of participant who has ever joined the meeting	0
have_guest	is there any participant other than the owner in the meeting ?	0
preparation_mode	is this a preparation mode meeting?	0

The meanings of each entry for "playback" are:

entry	meaning	default value
vid	virtual server id	0
session_index	playback session index	-1
name	player name	empty
playerid	player's internal id	empty
ip	IP address of the player as seen by the MCU	empty
elapsed_second	actual time passed since the start of the playback, in seconds	0
shared_playback	is this a shared playback?	0
end_reached	has the playback ever reached the end of file?	0
file_name	recording file name	empty
file_size	file size	0
file_duration	recording file duration in seconds	0
file_encrypted	is this file encrypted?	0

Because the HIT-enabled MCU supports both event-based meeting and person-based meeting, it is possible that one user has multiple online meetings at the same time. If the third party implements the interface for a guest to join an online meeting, it is important to use the same meeting type for the guest's join request as the online owner's. For example, if user "John" is in-meeting with an event-based meeting, the guest for this meeting should also use event-based meeting request. Otherwise, unexpected error may happen.

If the third party also supports the messenger interface, more information will be available in this set of status files. The additional online status information will be described in the following chapters.

The second set of status files adopt the name in the format of:

_office_status_ip_xxx.xml

The 'xxx' is the same as described above. Here is an example of a status XML file:

```
<?xml version="1.0" encoding="utf-8" ?>
<office_status>
   <user id="5" office_status="1" meeting_id="431"/>
   <user id="6" office_status="2" meeting_id="433"/>
</office_status>
```

This file lists all the open offices of the MCU server.

The meaning of ‘office_status” is as defined in the following table:

0	no active office
1	office is open, but no guest
2	office is open and at least one guest in the office
3	office is open, but the user is making a recording.

When there are multiple MCU servers in the system, the third party must use the _office_status_xxx.xml status files to make sure that all participants of the same meeting are assigned to the same MCU server. Specifically, when the third party prepares to create a jnj file for a person-based meeting, it should scan all the _office_status_xxx.xml status files to check whether an active office exists for the owner. If an active office is found at a particular MCU server, the jnj should be assigned to this MCU server.

3.5 Recording File Status File

The HIT-enabled MCU keeps a separate XML file for each meeting. The XML file is at the same directory of the recording file and has the same file name except the file extension is changed from “.jnr” to “.xml”. When the user requests a ‘meeting’ service without recording, the XML status file is still saved without the recording file. So this XML status file can be used to check the history of all meetings.

This XML file contains information such as meeting duration, participants, meeting title, which can be used by the third party to display additional details about a recording file. Here is an example of the status file:

```
<?xml version="1.0" encoding="utf-8" ?>
<status>
    <starttime utc="1477579954">20020426 15:51:00</starttime>
    <!-- duration is in seconds -->
    <duration>124</duration>
    <recording preparationmode=”1” read=”1086319845” shared=”1”
view_count=”3”>true</recording>
    <title>Test Recording</title>
    <!-- how the meeting is terminated. 0: empty room; 1: owner's
request; 2: timer is over; 3: server shutdown; 4: other reason; 5:
out of memory; 6: by admin -->
    <how>0</how>
    <participant userid="userid1" username="owner1" absent=”1”/>
    <participant userid="userid2" username="guest" />
</status>
```

The element “title” can be considered as comment to the recording file. The content of “title” is passed from the element “meetingtitle” of userinfo.

The element “how” shows how a meeting is terminated. A meeting can be terminated because:

- Empty room, i.e., all the participants leave the meeting room;
- The owner requests to terminate the meeting explicitly;
- Time is over. An event-based meeting has exceeded the scheduled duration;
- Server shutdown. The MCU server is shutdown and all active meetings are forced to be shutdown.

This element can be used to prevent a meeting to be launched multiple times. If element “how” is “1” or “2”, the meeting should not be launched again. If it is “0”, the situation becomes more complex. An empty room may be caused by that all the participants leave the room intentionally or they are disconnected by the network failures. In the latter case, the participants may still want to continue the unfinished meeting, although the server has terminated the meeting. So the meeting server may still need to launch a meeting for the users if the same meeting request comes again.

The element “recording” shows whether the meeting requests to be recorded. If element “recording” has an attribute “preparationmode”, then the corresponding recording file is the result of a preparation mode meeting. It is recommended that access for this recording file be restricted to delete only, while playback and download access should be disabled. If the element “recording” has an attribute “read”, it means that the corresponding recording file has been played back by its owner, i.e. the file is NOT ‘new’.

From MCU 3.9.0, the element “recording” has two new attributes: (1) “shared” means whether this file is shared, and (2) “view_count” shows how many times this file has been played back by others, excluding owner’s playback.

3.6 Participant Information File

When a meeting is active, the MCU keeps a participant information file in the subdirectory “participant” of the recording file. This file lists all the active participants in the current meeting. Based on this file, the third party can display the corresponding participant information if necessary.

A sample of the this file looks like this,

```
<?xml version="1.0" encoding="utf-8" ?>
<participant meetingid="241" guaranteed="0" utc="1477579954"
last_update="Sat Jun 05 00:02:28">
   <user name="a" userid="3" invited="1" ip=”192.168.1.100”/>
</participant>
```

The root tag is ‘participant’. Here, ‘participant’ has three attributes: ‘meetingid’, ‘guaranteed’ and ‘last_update’. Each subelement ‘user’ shows an active participant in this meeting.

3.7 Preparation Mode

The owner of a scheduled meeting, namely the meeting coordinator, can use a preparation-mode meeting to upload slides for an event-based meeting before the real meeting starts. Only a coordinator can start a preparation-mode event-based meeting. The third party can create a jnj file that assigns the attribute “preparationmode” to the element “command” for a coordinator. The coordinator uses this jnj file (e.g. “Prepare Meeting”) to start the preparation-mode meeting and prepare the uploaded slides.

The duration of a preparation-mode meeting is set to one hour and can be automatically extended. If a coordinator has terminated a preparation-mode meeting and want to prepare more slides, another preparation-session can be requested and the slides in the previous session will be transferred to the current preparation-session automatically. The MCU server deletes the recording file of the previous session if another preparation-session starts and the slides are transferred completely. All recording files for preparation-session consume the coordinator’s disk quota as well as a connection to a preparation-session consumes the service line.

Only un-deleted slides in the recording file of a preparation mode meeting are transferred to the normal meeting or the next preparation-session; while all other data including deleted slides, text chat, marks, etc. are discarded. The following scenario behaviors are defined for preparation-mode meeting:

Scenario	Behavior
A guest try to prepare a meeting	[This should never happen.] Get an error message “Illegal operation”
An invited guest tries to join a meeting while the meeting is in preparation mode	The coordinator will be prompted for a guest-join request. If the request is accepted, the meeting will be switched to normal mode; if the request is rejected or ignored, the guest will get an error message “The coordinator is preparing the meeting”. If the coordinator is not in the meeting, the guest enters the meeting room and the meeting is switched to normal mode.
A uninvited guest tries to join a meeting while the meeting is in preparation mode	The coordinator will be prompted for a guest-join request. If the request is accepted, the meeting will be switched into normal mode; if the request is rejected or ignored or the coordinator is not online, the guest get the same error message as in normal meeting mode.
A coordinator tries to prepare a meeting while the meeting is in preparation mode	The coordinator continues to prepare the meeting
A coordinator tries to join a meeting while the meeting is in preparation mode	The meeting will be switched to normal mode
A coordinator tries to prepare a meeting while the meeting is in normal mode	The coordinator get error message “The meeting has started, it’s too late to prepare it”
A coordinator tries to prepare a new meeting	If the normal mode meeting has been terminated, the coordinator gets error message “the session is over”; otherwise the coordinator starts a preparation mode meeting which is in preparation mode

Table 2. Preparation Mode Meeting Properties

When a meeting is switched from preparation mode to normal mode, all existing participants will receive a notification that the meeting is switched to the normal mode. The text chat and marks created during the preparation mode will be recorded if the preparation mode is switched to the normal mode. (Note that the information is discarded if the preparation mode meeting is terminated normally.)

The recording file for a preparation-session will NOT be deleted after the normal meeting starts and the slides in it are transferred into the normal meeting. So when the MCU server encounters a service break (e.g. crash, power down, etc.) and the normal meeting is restarted later, the prepared slides are still available.

It is recommended that the access to the recording file of a preparation-mode meeting be restricted to delete only, while playback and download access should be disabled. It is also recommended that the preparation-mode meeting should not be included in a meeting lookup operation.

When a coordinator is in a preparation-mode meeting, the online status for this coordinator will contains an attribute "preparationmode". The third party can decide to show this coordinator as in-meeting or off-meeting.

3.8 Server Backup

It is desirable to setup one or more backup meeting servers to continue providing meeting service in case the primary MCU is down or is put down for maintenance.

There are some requirements to set up the backup MCU servers:

- The recording directories of these MCU servers MUST be at the same location. This implies that a shared file system exists;
- All MCU servers MUST use the **same key pair**. This is to ensure that all the MCU servers can decrypt the jnj file created by the third party;
- The third party is aware of all the MCU servers.

When backup server is used, the third party needs to put one more entry to the created jnj file:

backupip=backupip1,backupip2,backupip3

If multiple backup servers are used, their IP addresses shall be separated by "," in the jnj file. Currently, at most 9 backup servers are supported. JoinNet will try the backup servers as the sequence in the jnj file if it cannot connect to the primary server.

There is another complementary entry in jnj file: "portm2". "portm2" specifies a secondary port that the servers are listening.

The JoinNet will try all the server/port combinations in the following orders:

primary_server/portm→primary_server/portm2→backup_server_1/portm→
backup_server_2/portm2→…→last_backup_server/portm→last_backup_server/portm2

Note, these connection attempts are parallel, and the time delay between two consecutive server/port connections is 500 milliseconds. The first successful connection out of all the combinations is adopted and all the other on-going connections will abort.

Note that, when the user specifies to use proxy and the portm2 is '443', the portm and portm2 will be switched (i.e., port 443 will be tried first when proxy is used).

3.9 Server Clustering & Balancing

A more advanced application of the server backup mentioned above is to put all the meeting servers into service at the same time. By this, all the meeting servers together can be viewed as a server cluster and load balancing can also be used among all the meeting servers.

The requirement of server clustering is the same as the server backup. However, to activate load balancing, the sequence of primary server and backup server need to be dynamic in each created jnj files.

We use a simple example to explain how to setup server cluster. Suppose there are three MCU servers at 192.168.1.1, 192.168.1.2, and 192.168.1.3. The recording directory is located at a separate storage server 192.168.1.10. All the MCU servers must set

```
recording=\\192.168.1.10\recording
```

in their configuration files (configm.ini).

The load balance is achieved by assigning different primary and backup server sequence in the jnj file for each different meeting. It is important to assign the ***same*** server sequence for all the jnj files of the *same* meeting (i.e. with the same "meetingid"); otherwise, the participants of a single meeting may connect to different servers and therefore cannot join the same meeting.

There are two ways to achieve load balance, one is statistical load balance, and the other is determinate load balance.

In a statistical load balance algorithm, the server sequence in the jnj files for each meeting is randomly assigned. For three meeting server, there are six choices of different server sequences. For example one of the sequences is:

```
ip=192.168.1.2
backupip=192.168.1.3,192.168.1.1
```

The third party does not need to know the status of the meeting servers in this statistical algorithm. The balance is achieved purely by statistics. Note if the capacity of each server (capacity information is in the license file of the server) is different, capacity-weighted algorithm is recommended.

In a determinate load balance algorithm, the third party knows the load of each server all the time. Whenever a new meeting is created, the best candidate server is calculated using the current load status and the known participants information in the new meeting. The third party is free to use any algorithm to achieve the load balance and this document does not give any example.

The load and aliveness status of each MCU server can be retrieved by reading the status files in the recording directory. Each server maintains two status files at the recording files, one is for aliveness, and the other is for load. Take server 192.168.1.1 as an example, the status file for aliveness is

```
_cluster_ip_192.168.1.1.txt
```

The status file for load is

```
_cluster_connection_192.168.1.1_.txt
```

Note that there is "_" after the IP address in the load status file. The aliveness status file contains "1" when the corresponding server is up; and contains "0" when it's down.

The load status file contains 14 (the last 4 are added since version 3.23.8) numbers separated by space in a single line representing:

> "meeting count" "max meetings supported" "connection count" "max connections supported" "outside connection count" "max outside connections supported" "reserved connection count" "max reserved connections supported" "reserved outside connection count" "max reserved outside connections supported" "playback connection count" "max playback connections supported" "outside playback connection count" "max outside playback connections supported"

For example, one of the servers may have (playback connections are separately maintained)

```
8 60 23 300 19 100 5 10 3 8 7 20 6 15
```

in its load status file, which means that

- ✓ The server has 8 meetings and 23 connections, out of which 19 connections are from outside;
- ✓ The server can support at most 60 meetings and 300 connections, out of which 100 connections can be from outside;
- ✓ The server has 5 reserved connections, out of which 3 connections are from outside;
- ✓ The server can support at most 10 reserved connections, out of which 8 connections can be from outside;
- ✓ The server has 7 playback connections, out of which 6 connections are from outside;
- ✓ The server can support at most 20 playback connections, out of which 15 connections can be from outside;

For a different example, one of the servers may have (playback connections are maintained as part of general connections)

```
8 60 23 300 19 100 5 10 3 8 7 0 6 0
```

in its load status file, which means that

- ✓ The server has 8 meetings and 23 connections, out of which 19 connections are from outside;
- ✓ The server can support at most 60 meetings and 300 connections, out of which 100 connections can be from outside;
- ✓ The server has 5 reserved connections, out of which 3 connections are from outside;
- ✓ The server can support at most 10 reserved connections, out of which 8 connections can be from outside;
- ✓ Out of the 23 connections, 7 are playback connections and the other 16(23 - 7) are meeting connections;
- ✓ Out of the 19 outside connections, 6 are playback connections and other 13(19 - 6) are meeting connections;

Whether the playback connections are maintained separately is controlled by license entries < max_playback_connection> and < max_playback_outconnection>.

When multiple meeting servers are organized as a cluster, the meeting server is not a SPOF (Single Point of Failure). However, there are other two SPOF: the third party and the storage server. When the third party is out of service, the end user cannot get fresh jnj file and cannot launch JoinNet. When the storage server is down, the end user cannot do playback or download, and the meeting will not be recorded (meeting function is still working; only recording fails).

3.10 Password Authentication

By default, HIT uses the freshness of the jnj file to authentication end-users. However, HIT also supports password authentication between the MCU server and end-users.

Because the third party has password information of its end users, so it can assign the base64 encoded SHA hash of the user's password to the element "password" of the userinfo in jnj file. When a end-user connects to the meeting server, the meeting server will ask for the password from the end-user. After the end-user input the password, the SHA hash of the inputted password is calculated and transmitted to the meeting server through the secure channel between the JoinNet and the meeting server. The meeting server can authenticate the end user by comparing the hash from the end-user and the hash from the third party.

The element "`timestamp`" in the userinfo specifies the expiration date of the password. The password will be disabled after this timestamp. The expiration date should not be too long for security reason. If an attacker intercepts the jnj file and knows the password from the other ways, he or she can masquerade the true user. A good expiration date (e.g. 1 day later) may avoid this type of attack.

If JoinNet supports cached old successful password input, end-users will not be prompted for password every time.

Because password authentication might prompt end-users to input their passwords, which is considered inconvenient, it is recommended only for confidential meetings.

Here are two examples for usage of password authentication.

> *Example #1: (User's account password is used in this example.) John invited Steven to a meeting and the meeting is considered confidential, so he sets the option at the third party's interface to use password authentication. The meeting invitation is sent to John and Steven through normal email. The meeting jnj file for John contains the hash of John's password while the jnj file for Steven contains the hash of Steven's password. These jnj files can be directly included in emails; so later when John and Steven can click the jnj files to launch JoinNet, John and Steven will be prompted for password to enter the meeting. The expiration date ('timestamp' attribute) of the password can be set as the end time of the meeting.*
>
> *Example #2: (User can specify a special entry code for a confidential meeting.) John invited Steven and Mike to a meeting and the meeting is considered confidential. John set the option through the third party's interface to assign a password for this meeting. The meeting invitation is sent to John, Steven and Mike through normal email. The meeting jnj files for all the three persons contain the hash of this password specified by John. These jnj files can be directly included in these emails, so later John Steven and Mike can click the jnj file to launch JoinNet and type in the required password. The expiration date ('timestamp' attribute) of the password can be set as the end time of the meeting.*

The third party is free to apply either example to use the password authentication or other types of secure applications. This password authentication is an optional feature. When convenience is more important, the third party can just use the default authentication, which relies on the freshness of the jnj file.

Note that, if the jnj file is delivered to the normal users through a secure channel such as HTTPS, the default authentication is also considered secure and can support confidential meeting.

3.11 The Third Party's Role

As far as HIT scheme is concerned, the third party is responsible for the following tasks:

- Manage and authenticate users. (All user can be managed by using database, simple text file or other technologies. The authentication is recommended to use secure one like SSL through web interface;)
- Generate jnj file for users for requesting service to the MCU;
- Rendering online and offline owner information to end-users;
- List owner's recording files and support download, delete from web. (Note that, when deleting the recording file, the corresponding status XML file is also to be deleted by the third party;)
- Generate a suitable jnj file that anyone can use if an owner wishes to publish one of his/her recording file to the public.

3.12 Recording File Archiving

MCU, version 3.10 and higher, supports recording file archiving, that is, once a recording file is archived by MCU, system administrator can retrieve a recording file even after the file is deleted by a user There are two settings in the configm.ini file that are related to this feature.

The first setting is "archive_recording_file". System administrator can set a non-zero value to this attribute to turn on the archiving feature. When a recording file is "deleted" by an owner, the file will not be removed from the MCU, but rather it is changed to a different file name. If "archive_recording_file" is set to a positive value, which means the days to archive a deleted recording file, the archived recording file will be removed by the MCU automatically after the configured time period. However, if the "archive_recording_file" is set to any negative value, the MCU will not automatically remove the archived files.

The second setting is "disable_jnr_password". When this setting is set to a non-zero value, the MCU will not create a password-protected recording file through meeting or changing JNR password. However, a user can still upload a password-protected recording file.

Before a recording file can be archived, its owner MUST be warned beforehand. Without being warned, the recording file should not be archived. Based on this rule, when the system turns on the archiving, the existing recording files will NOT be archived. Only the new recording file will be archived. When a recording file is created through meeting, all the JoinNet users in the meeting will be alerted that the recording file is going to be archived. Before a user uploads a recording file, the user should also be alerted that the

recording file is going to be archived. These recording files are tagged by using an attribute “to_archive” of the root element <status> in the corresponding recording status file as described in Section 3.5. For example:

```
<?xml version="1.0" encoding="utf-8" ?>
<status to_archive=”1”>
    <starttime>20020426 15:51:00</starttime>
    <!-- duration is in seconds -->
    <duration>124</duration>
    <recording preparationmode=”1” read=”1086319845” shared=”1”
view_count=”3”>true</recording>
    <title>Test Recording</title>
    <!-- how the meeting is terminated. 0: empty room; 1: owner’s
request; 2: time is over; 3: server shutdown; 4: other reason -->
    <how>0</how>
    <participant userid="userid1" username="owner1" absent=”1”/>
    <participant userid="userid2" username="guest" />
</status>
```

When archiving is turned on, recording feature will be enforced by the MCU, no matter the user requests recording or not. For this reason, it is better for the third party to enforce recording as well. Moreover, the JoinNet user cannot pause recording when archiving is on.

When a user deletes a recording file in which the corresponding status file contains the “to_archive” attribute, the file will be renamed instead of being deleted. For example, a file name of “test.jnr” will be renamed to “test.archive_jnr_xxxx”, where the “xxxx” is the time of deletion. The “xxxx” is the local time by using C library function time(). If a user tries to delete a recording file that is going to be archived through the third party’s interface, the third party should rename the file according to this rule.

When archiving is on, the third party is urged to alert the users when the user tries to upload recording file through the third party’s interface and the corresponding status file should contains the “to_archive” attribute.

3.13 One Time MCU Relay

From JoinNet 4.0, JoinNet supports entry “use_relay” in the jnj file. It is used by the MCU to specify a one-time MCU relay to be used for the JoinNet. The MCU relay information is not saved and will not affect other meetings.

The format of “use_relay” is

use_relay=mcu_relay,port

mcu_relay can be either domain name or IP address. Port information is required and comma is used to separate the relay part and port part.

4. JoinNet Messenger

HIT also supports JoinNet Messenger system, where the user's presence information can be managed. To support the messenger system, the MCU (version 3.0 or higher) acts as the messenger server, which maintains the users' messenger connections and sends meeting requests to the third party on the behalf of the messenger users. The request and response between the MCU and the third party is encrypted in the similar way as used for the authentication relay.

4.1 MCU's Configuration

There are two more settings in MCU's configm.ini for secure interaction between MCU and the third party.

```
messenger_web_url = http://mmc.homemeeting.com:8080/~webservice.php
messenger_dh_key = key_web_messenger_a.b.c.d_2333.x509
```

The messenger_web_url specifies the web url that the request will be sent. The messenger_dh_key specifies the public key of the third party used for encrypting the request and decrypting the response.

There is a new setting 'messenger_secure_channel 'in configm.ini from version 3.3.0. When this entry is set to 0, the request from the MCU server to the third party is not encrypted except that the request 'user authentication' is always encrypted. The third party must check the source IP address or other information to make sure the a request comes from an authentic MCU server.

4.2 Acquiring MCU's IP and Port

The third party's homepage should return the MCU's IP address and port information on a request

http://thirdparty.homepage.com/getmcuinfo/?userid=abcde

Otherwise, there will be no messenger service for this third party.

The MCU's info is written in xml format, e.g.,

```
<?xml version="1.0" encoding="utf-8" ?>
<mcuinfo>
    <ip>192.168.1.192</ip>
    <backupip>192.168.1.193</backupip>
    <port>443</port>
    <port2>2333</port2>
</mcuinfo>
```

4.3 Interaction Between MCU and The Third Party (Interaction initiated by MCU)

The interaction between MCU and the third party uses the authrelay algorithm similar to the one used to generate the jnj file.
The request from MCU is written in .xml format. Here is a preliminary example:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <disk_quota_alert internal_id="a"/>
</msgr_request>
```

This xml file is encrypted and signed and concatenated and a http request is sent as

```
http://messenger.web.url?param=key_mcu_192.168.1.63_0000|cIyqQrp7
VQdnvb6EN5pUvA3v90+43S59SbcRztr0EAg=|OukU+lvEeDaXyiLi7T7KJ+Iud4sd
vudnFDqN4/dzMDQ4y21pZtf38ds3yssSyODuCXlK+jbfmutqtflP9+HLHZmPe0Y7U
QAZdJUyo+twmqpwlys8i6bZAs5Z/N800dY7gN2YTGKpM4C2XbQOTooUQ1W9Tq5vMa
nz82Uin65NwjlY2X6aD1m1/4tXrRXpt84NxAfe0ZxAYOa4sohnV8u+KLbP3baQzry
HQYU/UE7VwFIkfZ2dD0NJlj2B6uL32ZrOIIMShGEyk/GMBjTEp9KlCYsYn8ne564S
wf6bwxK5K5zCkeJkyrn4WnchtJjoGQhP+2sPJY90cSn1MYFRrDH6FQ3tw/hvWZftl
WMjeBNEZJX9wDgF6TsBG5AHaORkbGa3t98=|n4xpLtt0GbkhSAr8dLA7iXK43gg=
```

The response from the third party is enclosed in the http response, which is similar to the userinfo2 in the jnj file.

When the path between the MCU and the Third Party is considered safe, the requests can be sent as plain text without encryption. This is controlled by entry “messenger_secure_channel” in configm.ini. When “messenger_secure_channel” is set to 0, the MCU will use plain text for the communication between the MCU and the Third Party. In this case, the web must check the Source IP of the requests to verify that the requests are from one of its authorized MCU servers. Otherwise, attackers can use unauthorized request to gain access to other persons’ files. The Third Party can use entry “authorizedIp” in mmc.ini to specify additional authorized IP addresses.

4.4 Request List

This section shows all the requests from the MCU to the third party and the corresponding responses. If the third party has problem for the keys (either it cannot decode the message, or it has difficulty to encode the response), it should return a HTTP non-200 response with detail error message as the content. Note that the errorcode contained in the response should be between 20000 and 29999. The error code is optional if detailed error text is included. The "public_status" in the response can be "1" or "0", where "1" means the user is public, i.e., everyone can see this user. The "sig" in the response shows the signature(such as hash of the content) for the content. If the value of the "sig" in the request matches the signature of the new response, it can be assumed that

the MCU already have the same content, so that the response can skip the content and set "no_change" to 1.

Several individual requests can be integrated in a batch request. Each individual request should have the attribute ‘seq’ to identify itself. The response for a batch request should keep the sequence order of requests. Partial response for batched request is allowed.

1) user authentication (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <user_auth seq=”0” userid="a" passwd="12345"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <user_auth seq=”0” passed="1" errorcode="20001" internal_id="a"
name="a" public_status="1">
    detailed error is written here if not passed
    </user_auth>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the ‘userid’ in the request is the userid for sign in purpose. The ‘internal_id’ in the response is the internal userid used by the third party to uniquely represent the user. Depending on different implementations of the third party, these two ‘id’ may be different, or may be the same. The "public_status" shows whether the user is visible to the public: a value of "1" means the user can be seen by everyone; while a value of "0" means only certain allowed users can see this user.
Note: if the authentication is successful, the MCU will cache the password and returned content for one hour.

2) user profile (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <user_profile seq=”0” internal_id="a"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <user_profile seq=”0” passed="1" errorcode="20001" name="a"
public_status="1">
    detailed error is written here if not passed
    </user_profile>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

3) query account info (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <query_info seq=”0” internal_id="a"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <query_info seq=”0” passed="1" errorcode="20001">
        detailed error is written here if not passed.
        <name>name</name>
        <email>email</email>
        <maxoutconnection>100</maxoutconnection>
        <diskquota used="27">1000</diskquota>
        <message new="3">7</message>
        <meeting new="11">24</meeting>
    </query_info>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

To the third party, a recording file can be determined as a message or a meeting according to the corresponding recording file's xml file. If the owner (first participant in the xml file) of the meeting is absent (attribute 'absent' is set for tag participant) in a recording file, it is a message; otherwise, it is a meeting.

To the third party, a message/meeting can be determined as new or not by checking the corresponding recording file's xml file. If the attribute 'read' is set for tag ‘recording’, it is not new; otherwise, it is new. The "<diskquota>" is in MB

4) launch office (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <launch_office internal_id="a" meeting_id=″meetingid″/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <launch_office passed="1" errorcode="20001">
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </launch_office>
</msgr_reponse>
```

Note: the meeting_id may be empty if the owner's office is not open yet.

5) owner_visit_owner (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <owner_visit_owner visitor_internal_id="visitor"
host_internal_id="host" meeting_id="meeting_id"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <owner_visit_owner passed=\1″ errorcode=\20001″>
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </owner_visit_owner>
</msgr_reponse>
```

6) guest_visit_owner (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <guest_visit_owner visitor_name="visitor" host_internal_id="host"
meeting_id="meeting_id"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <guest_visit_owner passed=\1″ errorcode=\20001″>
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </guest_visit_owner>
</msgr_reponse>
```

7) owner_leave_message (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <owner_leave_message visitor_internal_id="visitor"
host_internal_id="host"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <owner_leave_message passed="1" errorcode="20001">
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </owner_leave_message>
</msgr_reponse>
```

8) guest_leave_message (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <guest_leave_message visitor_name="visitor"
host_internal_id="host"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <guest_leave_message passed="1" errorcode="20001">
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </guest_leave_message>
</msgr_reponse>
```

9) owner_invite_guest (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <owner_invite_guest visitor_name="visitor" host_internal_id="host"
meeting_id="meeting_id"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <owner_invite_guest passed="1" errorcode="20001">
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </owner_invite_guest>
</msgr_reponse>
```

Note that the jnj for owner_invite_guest should set 'invited' to 1. Moreover, even if the owner has configured that s/he can only be viewable by her/his contacts, a jnj file should be generated for this request because it is an **explicit** invitation.

10) owner_invite_owner (MCU 3.0 or higher)

request xml:

```
<msgr_request>
    <owner_invite_guest visitor_id="visitor" host_internal_id="host"
meeting_id="meeting_id"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <owner_invite_owner passed="1" errorcode="20001">
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
    </owner_invite_owner>
</msgr_reponse>
```

11) allow list (MCU 3.3 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <allow_list seq="0" internal_id="a" sig="abcdefg"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <allow_list seq="0" passed="1" errorcode="20001" no_change="0"
sig="abcdefg" public_status="0">
        <i>a</i>
        <i>b</i>
        <i>c</i>
   </allow_list>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

This request queries the list of users that are allowed to see the provided internal_id. In the request, the attribute ‘sig’ is the signature received in the last response. If there is no change since last request, the web can just set ‘no_change’ to 1 and does not need to provide user list. Note the ‘public_status’ always needs to be set correctly. The ‘no_change’ should be set to ‘1’ if the user’s ‘public_status’ changes or his/her contact list changes. The subelements "<i>" show the internal_id of all users that are allowed to see the provided internal_id in the request.

12) meeting start notification (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <meeting_start_notification seq=”0” internal_id="a" meeting_id="m"
mcu_ip ="192.168.1.8" jnj_ip="mmc.homemeeting.com" utc=”123456789”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <meeting_start_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that mcu_ip specifies the IP address in decimal format, while the jnj_ip specifies the content configured in configm.ini. This notification tells the third party the MCU location for a specific meeting identified by the meetingid so that all the following participants can be directed to the same MCU. Here, ‘utc’ specifies the time that this request is sent by MCU.

13) meeting end notification (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <meeting_end_notification seq=”0” internal_id="a" meeting_id="m"
file_type ="message" visitor_name="v" utc=”123456789”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <meeting_end_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the attribute visitor_name is only meaningful when the recording file type is ‘message’ (the other type is ‘meeting’). Again, ‘utc’ specifies the time that this request is sent by MCU.

14) playback start notification (MCU 3.3 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <playback_start_notification seq=”0” internal_id="a" meeting_id="m"
file_name="" utc=”123456789” player_id=”a” player_name=”b”
ip=”192.168.1.1”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <playback_start_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the file_name has no extension (as specified in userinfo2.)

15) playback end notification (MCU 3.3 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <playback_end_notification seq=”0” internal_id="a" meeting_id="m"
file_name="" utc=”123456789” player_id=”a” player_name=”b”
duration=”222” data_received=”1” ip=”192.168.1.1”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <playback_end_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the file_name has no extension (as specified in userinfo2.) Here, ‘utc’ specifies the time that this request is sent by MCU; ‘duration’ is in ms, and ‘data_received’ shows whether a playback session receive any data from the server.

16) download start notification (MCU 3.3 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <download_start_notification seq=”0” internal_id="a" meeting_id="m"
file_name="" utc=”123456789” player_id=”a” player_name=”b”
ip=”192.168.1.1”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <download_start_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the file_name has no extension (as specified in userinfo2.)

17) download end notification (MCU 3.3 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <download_end_notification seq=”0” internal_id="a" meeting_id="m"
file_name="" utc=”123456789” player_id=”a” player_name=”b”
duration=”222” completed=”1” ip=”192.168.1.1”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <download_end_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the file_name has no extension (as specified in userinfo2.) Here, ‘utc’ specifies the time that this request is sent by MCU, ‘duration’ is in ms, and ‘completed’ shows whether the recording file is downloaded by the user completely.

18) disk quota alert (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <disk_quota_alert seq=”0” internal_id="a"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <disk_quota_alert seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

19) query web url (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <query_url internal_id="a" type=”homepage”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <query_url passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <url>base64 encoded url </url>
    </query_url>
</msgr_reponse>
```

Note that the type may be one of “homepage”, “checkmessage”, or “editprofile”. When the type is “checkmessage” and the internal_id is “blank” (i.e., this is a guest), the third party can decide to respond a blank url or the url pointing to the public recording files. The returned url for type “checkmessage” and “editprofile” should be volatile, i.e., it should be invalid after a time period, say 5 minutes.

20) public user list (MCU 3.0 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <public_user_list internal_id="a" hint="test" />
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <public_user_list passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <user internal_id=”1” userid=”a@a.com” username=”a”></user>
        <user internal_id=”2” userid=”b@b.com” username=”b”></user>
    </public_user>
</msgr_reponse>
```

Note that the returned users have preference to be publicly viewable. And the userid is the signin id, which may not be the same as the “internal_id”.

21) command list (MCU 3.3 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <command_list seq=”0”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <command_list seq=”0” passed="1" errorcode=”20001”>
        all supported commands are put here, in lower case, separated
by comma.
    </command_list>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

The MCU server uses this request to figure out what command the web supports. When the third party starts, it should notify the MCU to request the command list. The notification is sent through the interaction between the third party and the MCU (see section 4.8). Note that the comma is the only valid delimiter for the command list.

22) import contact (MCU 3.6 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
   <import_contact internal_id="abc"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <import_contact internal_id="abc" passed="1" errorcode="20001">
        <contactgroup groupname="c">
            <contact userid="a" username="aa"/>
            <contact userid="b" username="bb"/>
        </contactgroup>
    <import_contact/>
</msgr_reponse>
```

MCU use this request to ask for the contact group information of a specific user. Note the tag 'userid' in the <contact> means 'internal_id' here.

23) export contact (MCU 3.6 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <export_contact internal_id="abc">
        <contactgroup groupname="c">
            <contact userid="a"/>
            <contact userid="b"/>
        </contactgroup>
    </export_contact>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <export_contact internal_id="abc" passed="1" errorcode="20001">
    </export_contact>
</msgr_reponse>
```

MCU use this request to export contact group to the third party. Note the tag 'userid' in the <contact> means 'internal_id' here.

24) query event (MCU 3.6 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <query_event seq="0" sig="abc"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <query_event seq=”0” passed="1" errorcode=”20001” no_change="0"
sig="abcdefg">
        <event meeting_id="" title="" owner_id="" owner_name=""
            public_status="" start_time="" duration="" advance="">
            <user>abc</user>
            <user>def</user>
            <user>fff</user>
        </event>
    </query_event>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

This request queries the list of scheduled meetings/events. In the request, the attribute ‘sig’ is the signature received in the last response. If there is no change since last request, the web can just set ‘no_change’ to 1 and does not need to provide event list. The attribute ‘public_status’ specifies whether the public can see this event. When ‘public_status’ is set to ‘1’, all registered users can see this event. When ‘public_status’ is set to ‘2’, all users (include guests) can see this event. The attribute ‘start_time’ specifies the start time(the return value of C function time()) of a scheduled event. The attributes ‘duration’ specifies the duration (in seconds) of the event. The third party can set the ‘duration’ to 0 when the event duration is automatically extendable. The attribute ‘advance’ sets the advanced time (in seconds) that a member can join the event before the scheduled time. The entry ‘<user>’ lists all the invited registered member of this event. The content of ‘<user>’ shows the user’s internal id.

25) load client info (MCU 3.8 to MCU 3.14)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
   <load_ci internal_id="abc"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <load_ci internal_id="abc" passed="1" errorcode="20001">
        <ci office_name="Test">
           <contactgroup groupname="c">
               <contact userid="a" username="aa"/>
               <contact userid="b" username="bb"/>
           </contactgroup>
        </ci>
    <load_ci/>
</msgr_reponse>
```

MCU uses this request to load the client information of a specific user from the third party. Note the third party has saved the client information through a 'save_ci' request (see below) from MCU previously. If such information is not available, the third party should return an error.

From MCU 3.15, MCU directly save client info at recording folder. For example, client info of user "userid" is saved at file ~recording/_user/userid/_client_info_userid.dat.

26) save client information (MCU 3.8 to MCU 3.14)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <save_ci internal_id="abc">
        <ci office_name="Test">
           <contactgroup groupname="c">
               <contact userid="a" username="aa"/>
               <contact userid="b" username="bb"/>
           </contactgroup>
        </ci>
    </save_ci>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <save_ci internal_id="abc" passed="1" errorcode="20001">
    </save_ci>
</msgr_reponse>
```

MCU use this request to let the third party save the client information of a specific user. Note the third party does not need to understand the syntax of the client information. The content can be simply saved intact.

From MCU 3.15, MCU directly save client info at recording folder. For example, client info of user "userid" is saved at file ~recording/_user/userid/_client_info_userid.dat.

27) query_publish_file (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <query_publish_file seq="0" sig="abc"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <query_publish_file seq=”0” passed="1" errorcode=”20001”
no_change="0" sig="abcdefg">
        <publish_file blackbox_data="" title="" owner_name=""
            public_status="" duration="">
            <user>abc</user>
            <user>def</user>
            <user>fff</user>
        </publish_file>
    </query_publish_file>
</msgr_reponse>
```

Note the attribute 'seq' is optional, which is used in a batch request to identify the individual request. If 'seq' exists in the request, the attribute should be echoed intact in the response.

This request queries the list of published recording files. In the request, the attribute 'sig' is the signature received in the last response. If there is no change since last request, the web can just set 'no_change' to 1 and does not need to provide file list. The attribute 'public_status' specifies whether the public can see this published file. When 'public_status' is set to '1', all registered users can see this event. When 'public_status' is set to '2', all users (include guests) can see this event. The attribute 'blackbox_data' specifies a printable string that can be used to create the playback jnj when a user requests to playback this file. The attributes 'duration' specifies the duration (in minutes) of the file. The attribute 'title' sets the title of the recording file. The entry '<user>' lists all the members that can playback this file. The content of '<user>' shows the user's internal id.

28) upload file notification (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <upload_file_notification seq=”0” internal_id="a" meeting_id="m"
error ="" utc=”123456789” ip=”192.168.1.1”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <upload_file_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

Note that the error specifies the error code for the upload action. If the upload is successful, the error is ‘0’. ‘utc’ specifies the time that this request is sent by MCU.

29) upload file (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <upload_file internal_id="a"/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
     <upload_file passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <meetingid>test</meetingid>
     </upload_file>
</msgr_reponse>
```

This request asks for a meetingid that can be used to put the uploaded recording file. The meetingid can be created using the same criteria for creating a new meeting.

30) file status notification (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <file_status_notification seq=”0” internal_id="a" meeting_id="m"
file_name="" action=”” param=”” utc=”123456789”/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
    <file_status_notification seq=”0” passed="1" errorcode=”20001”/>
</msgr_reponse>
```

Note the attribute ‘seq’ is optional, which is used in a batch request to identify the individual request. If ‘seq’ exists in the request, the attribute should be echoed intact in the response.

MCU sends a notification to the third party when the MCU changes the status of a recording file. The ‘action’ specifies what change has been made. And the ‘param’ may be necessary to completely describe the change. The ‘action’ can be “archive”, “delete”, “mark_read” (non-zero param means the new status is read), “share”, “unshare”, “jnrpassword” (non-zero param means the file is password-protected), and “jnrtitle” (the ‘param’ show the new JNR title).

31) share file (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <share_file internal_id="a" meeting_id=”m” file_name=”f” expire=””
password=”” target_internal_id=””/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
     <share_file passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <url> base64 encoded url </url>
        <data> printable string </data>
     </share_file>
</msgr_reponse>
```

In the request, the ‘expire’ specifies the day number after which the shared url/data will be expired. If the ‘expire’ is 0, the share never expire. If the third party does not support expiration date, it should return an error. The ‘password’ specifies a password that the player must input to access the shared recording file. If the third party does not support password in a shared URL, it should return an error. An empty string for ‘password’

means there is no password restriction. The ‘target_internal_id’ specifies that only the user with the target internal_id can play this shared recording file. This information should be saved into the returned ‘data’ part. If the third party does not support this, it should return an error. For an anonymous share, the ‘target_internal_id” should be an empty string.

The ‘data’ in the response should be a printable string (use hex or base64 coding if the original data is binary). Its max length is 4096 (including the null at the end). This printable string will be provided to the third party when a user try to play this shared recording file.

For more information about sharing and unsharing a recording file, see section 4.9 below.

32) view_record (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <view_record internal_id="a" meeting_id=”m” file_name=”f” />
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
     <view_record passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <url> base64 encoded url </url>
     </view_record>
</msgr_reponse>
```

This request asks for a URL that lists the shared playback record for a specific file.

33) playback_file (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <playback_file internal_id="a" meeting_id=”m” file_name=”f” />
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
     <playback_file passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
     </playback_file>
</msgr_reponse>
```

This request asks for a jnj that can playback a specific recording file.

34) playback_share_file (MCU 3.9 or higher)

request xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_request>
    <playback_share_file internal_id="a" blackbox_data=””/>
</msgr_request>
```

response xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<msgr_response>
     <playback_share_file passed="1" errorcode=”20001”>
        detailed error is written here if not passed.
        <jnj>base64 encoded jnj file content</jnj>
     </playback_share_file>
</msgr_reponse>
```

This request asks for a jnj that can playback a shared or published recording file identified by the blackbox data.

4.5 Online Information for The Third Party

The MCU writes additional user online information to the status file _status_ip_xxx.xml, which is introduced in Section 3.4. The third party can use these additional information to show users' online status to its website.

A new tag <msgr_directory> is used for the additional online information. Here is an example:

```
<msgr_directory>
   <user name="a" userid="3" online="true" messenger_status="1"
office_status="0"/>
</msgr_directory>
```

Each ‘user’ specifies a user entry. The ‘name’ and ‘userid’ show the userid and username for the user. The tag ‘online’ has value either ‘true’ or ‘false’, which indicates whether the user is online. The detailed information is contained in the attribute ‘messenger_status’ and ‘office_status’. The meaning of ‘messenger_status’ is defined as in the following table:

1	online
2	busy
3	be right back
4	away
5	appear offline
6	mobile

The following online status calculation is recommended based on the combination of the ‘messenger_status’ and ‘office_status’:

- `if(messenger_status == ONLINE_STATUS_APPEAROFF)`
  `“Offline”;`
- `else if(messenger_status == ONLINE_STATUS_BUSY)`
  `“Busy”;`
- `else if(messenger_status == ONLINE_STATUS_RIGHTBACK)`
  `"Be Right Back";`
- `else if(messenger_status == ONLINE_STATUS_AWAY)`
  `"Away";`
- `else if(office_status == 2)`
  `"In Meeting";`
- `else if(office_status)`
  `"Office Open";`
- `else if(messenger_status == 0)`
  `"Offline";`
- `else if(messenger_status == ONLINE_STATUS_MOBILE)`
  `"Mobile";`
- `else`
  `"Online";`

4.6 Messenger Load Information

The messenger load information is maintained in status file

```
_cluster_messenger_connection_192.168.1.1_.txt
```

Note that there is “_” after the IP address in the load status file.

The load status file contains three numbers separated by space in a single line representing “messenger connection count”, “guest messenger connection count” and “maximum messenger connection count”.

For example, one of the servers may have

```
3563 24 5000
```

in its load status file, which means that the messenger server has 3563 messenger connections and 24 guest messenger connections while the maximum messenger connection number is 5000.

4.7 Between Messenger Server and Meeting Server

To be scaled up to serve a large number of users, different MCU servers can be designated to host the messenger services and meeting services, respectively. When the messenger-service MCU (i.e., messenger server) and the meeting-service MCU (i.e., meeting server) are used separately, they should use the same recording folder as specified in their configm files, since the messenger-service MCU needs to access the meeting status files written by the meeting-service MCU.

However, if it is not possible to configure the same recording folder for these two kinds of servers, the messenger-service MCU can adopt in an alternative way to access the meeting status maintained by the meeting-service MCU. In the configm file of a messenger-service MCU (, the setting 'meeting_server_ip' contains all the meeting server's IP addresses. A messenger server will initiate and maintain a TCP connection with each of these meeting servers. The meeting servers send the meeting status information through the TCP channel to the messenger servers. For security purpose, the meeting server can set the IP addresses ranges from where a messenger server can connect to it using the setting 'mmc_messenger_server_allow'.

If there is firewall between a messenger server and a meeting server, some proxy setting may be necessary to support the TCP connection.

4.8 Interaction Between Third Party and Messenger Server (Interaction initiated by third party)

From MCU version 3.6.0, the third party can send the messenger server some notification through the messenger server's internal web service interface.

A messenger server, i.e., messenger-service MCU, supports an internal web service at **http://mcuip:mcuport/**

When the third party starts, it should visit **http://mcuip:mcuport/web2mcu/init** (if possible) so that the messenger server will be triggered to send an 'command list' request to the third party. This is requirement is optional as the messenger server periodically

requests the ‘command list’ once a day just in case the third party does not support this visit.

When there is any scheduled event change, including start time, duration, or participant change, the third party should visit **http://mcuip:mcuport/web2mcu/event** and the messenger server will be triggered to send an ‘event update’ request to the third party.

When there is any published recording file change, the third party should visit **http://mcuip:mcuport/web2mcu/publish_file** and the messenger server will be triggered to send an ‘publish file update’ request to the third party.

When a user changes his preference (such as whether the public can see him/her), the third party should visit **http://mcuip:mcuport/web2mcu/preference?internal_id=abc** and the messenger server will be triggered to retrieve the updated preference information for the user with internal ID ‘abc’.

When a user changes his name, the third party should visit **http://mcuip:mcuport/web2mcu/user_update?internal_id=abc** and the messenger server will be triggered to retrieve the updated name information for the user with internal ID ‘abc’.

Note, when there are multiple messenger servers in the system, the third party should visit the corresponding links of all the messenger servers.

4.9 Share and Unshare Recording Files

From MCU version 3.9.0, the MCU supports sharing and unsharing a recording file. To be compatible with the MCU, the third party needs to meet the following requirement. Otherwise, the third party should return an error when a user requests to share or unshare a file.

1. When a user requests to playback a shared recording file, but the corresponding .xml file's 'shared' flag is reset (see section 3.5), the third party should return the user an error.
2. When the third party shares a file, it should set the ‘shared’ flag in the corresponding .xml file.
3. When the third party unshares a file, it should reset the ‘shared’ flag in the corresponding .xml file.

4.10 MCU Clustering

From MCU version 3.12.0, the MCU supports clustering, i.e., multiple (up to 32) MCU servers can be clustered together to support large number (up to 100,000) of messenger clients. To setup MCU clustering, please follow the following instructions.

The clustered MCU server requires a license that enables 'clustering'. All MCU servers must use the same 'recording' folder and use the same key pair (see Section 3.8). Create a text file 'msl.txt' in the recording folder that lists all the clustered MCU server's IP addresses: one IP per line. In the configm.ini file of each MCU server, add one entry 'self_messenger_server_ip' to tell the MCU which IP belongs to it. This IP should be one of the lines in the file msl.txt. Here is an example of an msl.txt file:

```
192.168.1.8
192.168.1.119
```

Note if you are using local-range IPv6 addresses that need the interface part of the address, copy the msl.txt file to the executable folder (where configm.ini is) so that you can append interface part to the IPv6 address. When MCU finds an msl.txt in the executable folder, the msl.txt file in the recording folder is ignored. Interface part is not allowed for the IPv6 addresses in the msl.txt file located in the recording folder. For example, if MCU1 uses IP fe80::213:2ff:fe20:2454 and MCU2 uses IP fe80::213:2ff:fe20:2455, the msl.txt file in MCU1's folder could be

```
fe80::213:2ff:fe20:2454%2
fe80::213:2ff:fe20:2455%7
```

while the msl.txt file in MCU2's folder could be

```
fe80::213:2ff:fe20:2454%5
fe80::213:2ff:fe20:2455%9
```

There is no encryption for the communication among the clustered MCU servers for performance reason. The MCU server MUST be able to connect to each other directly without any proxy setting.

The third party can control which MCU server a messenger client connect to by controlling the response of the client's getmcuinfo request (See Section 4.2). If the third party decides to use load balance strategy to decide the MCU assignment, it can read the server's status file to determine the server's load (See Section 4.6).