Dialogic® PowerMedia™ HMP for Linux Release 4.1

Release Notes

December 2023 05-2680-044

www.dialogic.com

Copyright and Legal Notice

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Enghouse and its affiliates or subsidiaries ("Enghouse"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Enghouse does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND ENGHOUSE, ENGHOUSE ASSUMES NO LIABILITY WHATSOEVER, AND ENGHOUSE DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF ENGHOUSE PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Enghouse products are not intended for use in certain safety-affecting situations.

Due to differing national regulations and approval requirements, certain Enghouse products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Enghouse at legal.operations@enghouse.com

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Enghouse may infringe one or more patents or other intellectual property rights owned by third parties. Enghouse does not provide any intellectual property licenses with the sale of Enghouse products other than a license to use such product in accordance with intellectual property owned or validly licensed by Enghouse and no such licenses are provided except pursuant to a signed agreement with Enghouse. More detailed information about such intellectual property is available from Enghouse's legal department at **80 Tiverton Court, Suite 800 Markham, Ontario L3R 0G4**.

Enghouse encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Dialogic, Dialogic Pro, DialogicOne, Dialogic Buzz, Brooktrout, BorderNet, PowerMedia, PowerVille, PowerNova, ControlSwitch, I-Gate, Veraz, Cantata, TruFax, and NMS Communications, among others as well as related logos, are either registered trademarks or trademarks of Enghouse and its affiliates or subsidiaries. Enghouse's trademarks may be used publicly only with permission from Enghouse. Such permission may only be granted by Enghouse legal department at **80 Tiverton Court, Suite 800 Markham, Ontario L3R 0G4**. Any authorized use of Enghouse's trademarks will be subject to full respect of the trademark guidelines published by Enghouse from time to time and any use of Enghouse's trademarks requires proper acknowledgement.

The names of actual companies and products mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Enghouse is not responsible for your decision to use open source in connection with Enghouse products (including without limitation those referred to herein), nor is Enghouse responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Table of Contents

1. Revision History

This section summarizes the changes made in this and, if applicable, each previously published version of the Release Notes for PowerMedia HMP for Linux Release 4.1, which is a document that is planned to be periodically updated throughout the lifetime of the release.

Revision	Release Date	Notes
05-2680-044 (updated)	October 2023	Updates to support PowerMedia HMP 4.1 Linux SU 422. Post Release Developments: Support for MKV and WebM file formats Release Issues: HMP-1716, HMP-1715, HMP-1695, HMP-1694, HMP-1676, HMP-1664
05-2680-043	June 2023	Updates to support PowerMedia HMP 4.1 Linux SU 421. Post Release Developments: Added new feature support for Linux distributions, SE Linux support, and first party call control with SRTP. Release Issues: HMP-1625, HMP-1596, HMP-1500
05-2680-042	December 2022	Updates to support PowerMedia HMP 4.1 Linux SU 420. Added NAT Traversal feature description to Section 4. Release Issues: HMP-1559, HMP-1547, HMP-1527, HMP-1512
05-2680-040	October 2021	Archived previous revision of the document. Updates to support PowerMedia HMP 4.1 Linux SU 413. Release Issues: Added the following resolved defects: HMP-1331, HMP-1311, HMP-1294, HMP-1223.

2. Post Release Developments

This section describes significant changes after the general availability release.

Support for MKV and WebM file formats

With Service Update 422, HMP for Linux now supports MKV and WebM file formats through mm_Play() and mm_Record() APIs.

For more information, refer to the Documentation Updates section later in this document.

AlmaLinux 8.7 and Oracle Linux 8.7 Support

With Service Update 421, HMP for Linux now supports AlmaLinux 8.7 (and later) and Oracle Linux 8.7.

Support for SELinux

Service Update 421 adds support for SELinux in IP only configurations. The SELinux targeted policy can now be enabled on systems running HMP 4.1 SU421 or later.

Care should be taken when enabling SELinux from a disabled state since improper configuration may prevent the system from starting. It is recommended setting the SELinux policy to targeted and the mode to permissive before enforcing the policy. Changing the mode to enforcing should only be done after resolving any issues logged while in permissive mode. Please refer to the RedHat documentation on the proper steps for enabling SELinux and enforcing the targeted policy.

Once SELinux is enabled and enforcing the targeted policy on the host operating system, the HMP 4.1 software can be installed and configured as normal.

Configurations using DNI boards are currently not supported. The SELinux Multi-Level Security policy is not supported.

Secure RTP With First Party Call Control

With Service Update 421, HMP for Linux adds functionality to support Secure RTP calls when using the first party call control (1PCC) model.

For more information, refer to the Documentation Updates section later in this document.

File Location Changes in HMP Linux Service Update 420

With Service Update 420, the locations of temporary and internal files used for communication between HMP components have been consolidated. Internal files created by RTF (excluding log files), device management, OA&M, and host runtime libraries have been relocated under the "/run/dialogic" directory. Files under this directory are removed when HMP services are stopped or the HMP Linux build is uninstalled. The HMP Linux installation directory, configuration, and log file locations remain unchanged. No changes are required to existing customer applications.

NAT Traversal Support for 1PCC Applications in Cloud Environments

With Service Update 420, HMP for Linux adds NAT Traversal functionality that enables SIP and RTP access to the public network in cloud environments where a media server only has access to a network interface with a local IP address.

For more information, refer to the Documentation Updates section later in this document.

Transmit RFC 2833/RFC 4733 Tone Events

With Service Update 417, HMP for Linux adds support for sending RFC 2833/RFC 4733 telephony events. This feature allows an application to transmit a sequence of both DTMF and non-DTMF telephony events, including hookflash, over an IP network.

For more information, refer to the Documentation Updates section later in this document.

Red Hat Enterprise Linux (RHEL) 8 and Rocky Linux 8.4 Support

With Service Update 413, Dialogic® PowerMedia™ HMP for Linux Release 4.1 now supports Red Hat Enterprise Linux (RHEL) 8.1 or later and Rocky Linux 8.4 or later.

Before you proceed with installing PowerMedia HMP Linux, ensure that you install the following Red Hat packages:

- Development Tools packages
- libstdc++.i686
- glibc.i686
- libxcrypt.i686
- libnsl.i686
- libnsl.x86_64
- nss-util.i686
- elfutils-devel.x86_64 (required when installing Dialogic® HMP Interface Boards)
- compat-openssl.10.1686 (required if running 32 bit application)
- libuuid.i686

3. Release Issues

Issue Type	Defect No.	SU No.	Product or Com-ponent	Description
Resolved	HMP-1716	422	SSP	An update was made to address a memory leak when using NbUP protocol.
Resolved	HMP-1715	422	IPHOST	An issue introduced in SU421 that impacted SIP call rates/density has been addressed.
Resolved	HMP-1695	422	IPHOST	An update was made to prevent additional characters from being included in the 200 OK SDP a=crypto line.
Resolved	HMP-1694	422	IPHOST	An update was made to address an IPHOST library exception when accepting a ReINVITE during a 1PCC SRTP call.
Resolved	HMP-1676	422	Install	An update was made to dlservices to resolve an issue when configuring ports.
Resolved	HMP-1664	422	IPHOST	An issue that caused SIP TLS connections to fail has been addressed.
Resolved	HMP-1625	421	IPHOST	An issue was addressed that prevented use of Elliptic Curve Diffie-Hellman ciphers for SIP TLS.
Resolved	HMP-1596	421	DIAG	An issue that caused corruption in RTF logs when the process id was larger than 6 digits has been addressed.
Resolved	HMP-1500	421	IPHOST	An update was made to ensure that SIP INVITE messages sent after a CANCEL contain the proper SDP.
Resolved	HMP-1559	420	SSP	An update was made to relocate internal files used for communication between HMP components from /tmp/dialogic to /run/dialogic to resolve SSP errors caused by the systemd-tmpfiles service.
Resolved	HMP-1547	420	IPHOST	An issue that caused a SIP INVITE to be sent without SDP after a CANCEL has been addressed.

Issue Type	Defect No.	SU No.	Product or Com-ponent	Description
Resolved	HMP-1527	420	SSP	An issue that caused a memory leak in the SSP process has been addressed.
Resolved	HMP-1512	420	Driver	An update was made to resolve a driver build issue during installation on RedHat 8.6 and Rocky 8.5.
Resolved	HMP-1495	417	SSP	An issue was addressed that caused lost audio between DX or thinblade timeslot devices and MCX conference party devices.
Resolved	HMP-1480	417	SSP	An update was made to reduce error/warning messages when processing RTCP packets.
Resolved	HMP-1479	417	SSP	An issue was addressed that prevented setting AMR codec options when calling ipm_ModifyMedia().
Resolved	HMP-1478	417	SSP	An update was made to allow codec updates and IP address and port changes when calling ipm_ModifyMedia().
Resolved	HMP-1466	417	IPHOST	An issue that caused a crash during SIP stack initialization has been addressed.
Resolved	HMP-1459	417	OAM	An issue that caused PDKManager to fail has been addressed.
Resolved	HMP-1331	413	SSP	A memory leak in the SSP component when using the NbUP with AMR codec was addressed.
Resolved	HMP-1311	413	IPHOST	An issue that caused SIP INVITE to be rejected with 400 “Sip Parser Error” for certain SIP “Identity” header formats was addressed.
Resolved	HMP-1294	413	Driver	The issue that caused the Linux kernel to crash while starting HMP thinblade Services was addressed.
Resolved	HMP-1223	413	IPHOST	The issue that caused incorrect handling of the SIP REINVITE message when multiple RFC2833 lines are present in the SDP was fixed.

4. Documentation Updates

NAT Traversal Feature

NAT Traversal functionality enables SIP and RTP access to the public network in cloud environments where a media server only has access to a network interface with a local IP address.

SIP and SDP address translation is configured on gc_Start() by setting the "nat_external_sip_address" and "nat_external_rtp_address" fields in the IP_VIRTBOARD structure.

The SIP external address is used to replace the host part of the addresses in the "From" header and the top "Via" header in outbound SIP request messages. The host part of the address is replaced in the "Contact" header in outbound SIP request messages and outbound SIP response messages. The SIP external address is used in 1PCC and 3PCC operating modes.

An application can use IPSET_SIP_MSGINFO / IPPARM_SIP_HDR to add SIP headers. The application must translate addresses for header types that aren't known to GlobalCall.

The RTP external address is used to replace the host part of the addresses on the o= and c= lines in outbound SDP. The RTP external address is used in 1PCC operating mode only.

The "audio_rtp_base_port" field in the IP_VIRTBOARD structure is used to configure unique UDP port ranges for the IPM devices in 1PCC operating mode when multiple media servers share one public IP address.

In a cloud environment where the media server has a local IP address, only the SIP and SDP external addresses must be configured. The SIP and SDP external addresses are set in the "nat_external_sip_address" and "nat_external_rtp_address" fields of the IP_VIRTBOARD structure.

IP_VIRTBOARD Additions for NAT traversal

The following parameters have been added to the IP_VIRTBOARD structure to support NAT Traversal feature. For more information regarding the IP_VIRBOARD structure, refer to the *Dialogic® Global Call IP Technology Guide*.

nat_external_sip_address (structure version >= 0x118 only)
: Specifies the host address that will replace the host address in From, Contact and Via headers in outbound SIP messages. The value can be any string, e.g. an IPv4 address, an IPv6 address or an FQDN. SIP address translation is disabled by default. This field applies to 1PCC and 3PCC operating modes.

nat_external_rtp_address (structure version >= 0x118 only)
: Specifies the host address that will replace the host addresses on the c= and o= SDP lines in all outbound SDP. The value must be an IPv4 address or an IPv6 address. SDP address translation is disabled by default. This field applies to 1PCC operating mode only.

audio_rtp_base_port (structure version >= 0x118 only)
: Sets the IPM base UDP port. The default value is 0 which means the default IPM base UDP port will be used. This field applies to 1PCC operating mode only.

Configuring Multiple Servers Sharing a Single Public Address

Multiple media servers can also share a single public IP address. Forwarding rules are configured on the NAT device for each media server. Each media server's UDP and TCP ports are configured so they don't overlap.

Media server 1 configuration

SIP UDP port 5060

SIP TCP port 5060

RTP / RTP base UDP port 20000

Media server 2 configuration

SIP UDP port 5070

SIP TCP port 5070

RTP / RTP base UDP port 30000

SIP and RTP to media server 1

Public		Private
SIP 172.1.1.10:5060	->	192.168.1.20:5060
RTP 172.1.1.10:20000	->	192.168.1.20:20000

SIP and RTP to media server 2

Public		Private
SIP 172.1.1.10:5070	->	192.168.1.30:5070
RTP 172.1.1.10:30000	->	192.168.1.30:30000

The following IP_VIRTBOARD fields are used to configure the network interface IP address, UDP ports and TCP port for SIP on gc_Start():

localIP

localIPv6

localIPv6_iface_name

sip_signaling_port

audio_rtp_port_base

Transmit RFC 2833/RFC 4733 Tone Events

This feature allows an application to transmit a sequence of both DTMF and non-DTMF telephony events over an IP network by calling ipm_SendTelephonySignals( ). This can be used in generating all RFC 4733 tone event definitions (0-255) beyond the initial set of DTMF telephony events (0-15) used to represent digits 0-9, A-D, *, #. This can also be used to generate a non-DTMF telephony event, such as a Hookflash event, and DTMF RFC 2833/RFC 4733 RTP telephony events based on WebRTC signaling events in a WebRTC Gateway application. The feature also allows an application to support the modem and text tone event definitions specified in RFC 4734 (https://tools.ietf.org/html/rfc4734), or channel oriented signaling tone events specified in RFC 5244 (https://tools.ietf.org/html/rfc5244).

The RFC 4733 (https://tools.ietf.org/html/rfc4733) recommendation specifies the "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals" and obsoletes the original RFC 2833 specification. The send/receive RFC 4733 tone event capability is integrated into the standard DTMF digit generation and detection API when RFC 2833/RFC 4733 mode is negotiated upon SDP media session establishment.

Note: ipm_SendTelephonySignals() is only available when audio is encoded. It's not available for native audio.

New DTMF Transfer Mode

The Dialogic® IP Media Library API can be used to configure which DTMF transfer mode (in-band, RFC 2833, or out-of-band) is used by the application. The mode is set on a per-channel basis using ipm_SetParm( ) and the IPM_PARM_INFO data structure.

DTMFXFERMODE_RFC2833_APP

This transfer mode is the same as DTMFXERMODE_RFC2833 except that inband tones are not converted into RFC 2833 DTMF events by the IPM transmitter. The tones are still clamped. In this mode, telephony events are generated exclusively by the application when ipm_SendTelephonySignals() is called.

Function Information

Name: int ipm_SendTelephonySignals (nDeviceHandle, *pDigitInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle IPM_TELEPHONY_SEQUENCE_INFO
*pInfo	• pointer to information structure
unsigned short usMode	• async or sync mode setting

Returns:

0 on success

-1 on failure

Includes:

srllib.h, ipmlib.h

Category:

Media Session

Mode: asynchronous or synchronous

Description

The ipm_SendTelephonySignals( ) function instructs the IPM device to generate a sequence of RFC 2833/RFC 4733 telephony events over an IP network. The on/off time and volume of each telephony event is configurable.

The transfer mode must be set to DTMFXERMODE_RFC2833 or DTMFXERMODE_RFC2833_APP for the telephony events to be transmitted on the network. Refer to the ipm_SetParm( ) for more information.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pInfo	pointer to the IPM_TELEPHONY_SEQUENCE_INFO structure
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution

Termination Events

IPMEV_SEND_TELEPHONY_SIGNALS

Indicates successful completion. The given telephony event sequence has been transmitted to the remote endpoint. If ipm_Stop( ) is called while a sequence is being generated, generation is stopped immediately and the IPMEV_SEND_TELEPHONY_SIGNALS termination event is generated, followed by the IPMEV_STOP termination event.

IPMEV_SEND_TELEPHONY_SIGNALS_FAIL

Indicates that the function failed. See the "Errors" section below for a list of error codes.

Cautions

ipm_SendTelephonySignals() is only available when audio is encoded. It’s not available for native audio.

When the transfer mode is set to DTMFXFERMODE_RFC2833, inband tones that are converted to telephony events will conflict with telephony events that are generated by the

application at the same time.

The DTMFXFERMODE_RFC2833_APP mode disables telephony event generation from inband tones on the transmit side. While in DTMFXFERMODE_RFC2833_APP mode, an application can detect inbound tones or telephony events using a DX device or detect inbound telephony events using IPM telephony event reporting. The detected tones/events can be regenerated using ipm_SendTelephonySignals( ).

Errors

If the function returns -1 to indicate failure, call ATDV_LASTERR( ) and ATDV_ERRMSGP( ) to return one of the following errors:

EIPM_BUSY

Channel is busy.

EIPM_INTERNAL

Internal error.

EIPM_INV_MODE

Invalid mode.

EIPM_INV_STATE

Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM

System error.

If the IPMEV_SEND_TELEPHONY_SIGNALS_FAIL termination event is received, call ATDV_LASTERR( ) and ATDV_ERRMSGP( ) to return one of the following errors:

EIPM_INVALID_EVENT_ID

The event ID is not in the range 0 through 255.

EIPM_INVALID_VOLUME

The volume is not in the range 0 through 63.

EIPM_INVALID_OPTIONS

Options value is wrong.

EIPM_INVALID_SIGNAL_TYPE

Unrecognized signal type. The signal type must be "event".

EIPM_PAYLOAD_TYPE_NOT_IMPLEMENTED

Per sequence/per event RTP payload type is not implemented. The telephony event payload type must be set by the PARMCH_RFC2833EVT_TX_PLT to ipm_SetParm( ).

EIPM_PTIME_NOT_IMPLEMENTED

Per sequence/per event ptime is not implemented. The frame time of the codec selected in ipm_StartMedia( ) is always used.

EIPM_CLOCK_RATE_NOT_IMPLEMENTED

Per sequence/per event clock rate is not implemented. The RTP clock rate of the codec selected in ipm_StartMedia( ) is always used.

EIPM_TONE_NOT_IMPLEMENTED

This function can only generate telephony events. Telephony tones can not be generated by this function.

EIPM_OUT_OF_RANGE

An event ID in the "telephony event ID string" is too large. The maximum value is 255.

EIPM_TOO_MANY_DIGITS

An event ID in the "telephony event ID string" contains too many digits. The maximum number of digits is 3.

EIPM_INVALID_CHARACTER

The "telephony event ID string" contains an invalid character. Valid characters are comma and decimal digits.

Example 1

In this example, the "telephony event ID string" is used to generate the sequence.

```
void SendTelephonySignalsExample1(int handle)
{
     IPM_TELEPHONY_SEQUENCE_INFO seq;
     //    The second parameter to the INIT function is set to zero since it
     doesn't apply
     //    to strEventIDs. It's only used for the signal array.
     INIT_IPM_TELEPHONY_SEQUENCE_INFO(&seq, 0);
     // These apply to each event in the event ID string.
     seq.sVolume = 7; // dBm0/sign dropped as defined in RFC 4733 seq.usDuration =
     200; // milliseconds
     //    This is the duration of the gap between events in the event ID string.
     seq.unInterval = 100; // milliseconds
     //    The event ID string is a comma separated list of telephony event IDs.
     The following
     //    list includes DTMF 1,2,3 and hook flash.
     seq.strEventIDs = "1,2,3,16";
     if( ipm_SendTelephonySignals(handle, &seq, EV_ASYNC) == -1 )
     {
     printf("ipm_SendTelephonySignals() failed, %s (%ld)\n", ATDV_ERRMSGP(handle),
     ATDV_LASTERR(handle));
```

```
    }

    FREE_IPM_TELEPHONY_SEQUENCE_INFO(&seq); // this must be called
}
```

Example 2

In this example, the "telephony signal array" is used to generate the sequence. The volume and duration can be configured for each event.

```
void SendTelephonySignalsExample2(int handle)
{
    IPM_TELEPHONY_SEQUENCE_INFO seq;

    IPM_TELEPHONY_EVENT_INFO      pEventInfo;

    //    The "count" parameter is set to 3 since there are 3 elements in the
    telephony

    //    event array defined below.

    INIT_IPM_TELEPHONY_SEQUENCE_INFO(&seq, 3);

    // These are the default values that will be used for the event array
    elements.

    seq.sVolume = 7;

    seq.usDuration    = 200;

    seq.unInterval    = 100;

    //    The volume and duration aren't set in this element, so the default
    values

    //    above are used.

    pEventInfo = INIT_IPM_TELEPHONY_EVENT_INFO(&seq);

    pEventInfo->eTelephonyEventID = SIGNAL_ID_EVENT_DTMF_0;

    //    The volume and duration set on the next two events override the volume,
    duration

    //    and interval values set above.

    pEventInfo = INIT_IPM_TELEPHONY_EVENT_INFO(&seq);

    pEventInfo->eTelephonyEventID = SIGNAL_ID_OFF; // pseudo event ID to insert a
    gap between events

    pEventInfo->usDuration  = 150;

    pEventInfo = INIT_IPM_TELEPHONY_EVENT_INFO(&seq);

    pEventInfo->eTelephonyEventID = SIGNAL_ID_EVENT_DTMF_1;

    pEventInfo->sVolume     = 5;

    pEventInfo->usDuration = 185; // this is rounded up to the next frame period,
    e.g. 200 milliseconds for 20 millisecond G.711

    if( ipm_SendTelephonySignals(handle_, &seq, EV_ASYNC) == -1 )

    {
```

```
        printf("ipm_SendTelephonySignals() failed, %s (%ld)\n", ATDV_ERRMSGP(handle_),
        ATDV_LASTERR(handle_));

        }

        FREE_IPM_TELEPHONY_SEQUENCE_INFO(&seq); // this must be called
    }
```

For more information on IP Media API functionality, refer to the Dialogic® IP Media Library API Programming Guide and Library Reference.

Secure RTP With First Party Call Control

Secure RTP (http://www.ietf.org/rfc/rfc3711.txt) is a method that allows for secure encrypted transmission of RTP data between endpoints. Secure RTP functionality has been previously supported with HMP when using third party call control (3PCC) mode. In 3PCC mode, the application is responsible for selecting the encryption method, key generation, negotiation, and state transitions between the endpoints. This HMP release adds functionality for supporting Secure RTP when using first party call control (1PCC) configuration. In 1PCC mode, these steps are managed within the HMP GlobalCall libraries. Secure RTP can be used in conjunction with SIP TLS (https://www.rfc-editor.org/rfc/rfc5246.txt) to provide a secure method for two endpoints using SRTP to exchange the necessary setup information, including SRTP keys.

Supported crypto suites

HMP supports the following crypto suites through 1PCC:

- AES_CM_128_HMAC_SHA1_80
- AES_CM_128_HMAC_SHA1_32
- AES_CM_256_HMAC_SHA1_80
- AES_CM_256_HMAC_SHA1_32

Enabling 1PCC SRTP

The feature is enabled by including a GC_PARM_BLK with parameter set ID GCSET_CHAN_CAPABILITY and parameter ID of IPPARM_ENABLE_SRTP_1PCC. Setting the value to IP_DISABLE disables Secure RTP (default), setting to IP_ENABLE enables Secure RTP.

The HMP license on the target system must include Encryption (SRTP / TLS) in the license configuration to enable support for this feature. An "IPERR_BAD_PARAM" is returned when enabling the feature if Encryption is not included in the HMP license.

Example code

…

```
    /* Enable Secure RTP */
    gc_util_insert_parm_val(&gcParmBlk, GCSET_CHAN_CAPABILITY,IPPARM_ENABLE_SRTP_1PCC,
     sizeof(long), IP_ENABLE);
    if (gc_SetUserInfo(GCTGT_GCLIB_CHAN, lineDev , gcParmBlk, GC_ALLCALLS) < 0) {
       printf("Error: gc_SetUserInfo() returned error enabling Secure RTP\n");
```

```
}
```

Outbound calls

When the SRTP feature is enabled, the GlobalCall library will generate a new Master key for each call. The crypto information is included in the SDP offer provided in the SIP INVITE message. The remote side responds with its crypto information in its response. If the negotiation is successful, RTP in both directions will be encrypted. If the remote side does not provide its crypto information in its response, the call will be rejected per RFC4568, section 7.1.2 (https://www.ietf.org/rfc/rfc4568.txt)

The table below shows documents HMP behavior during outbound call negotiation.

Local	Remote response	Results
SRTP Disabled	SDP without SRTP	Invoke regular RTP call handling
SRTP Enabled	SDP with SRTP	Invoke Secure RTP call handling
SRTP Disabled	SDP with SRTP	If remote responds with SRTP crypto, then local side will CANCEL the call and return GCEV_DISCONNECTED event sent to application with reason IPEC_InternalReasonSRTPCryptoMismatch.
SRTP Enabled	SDP without SRTP	Per RFC, if the remote does not support SRTP then it should reject the call. If it accepts without sending SRTP crypto back then local side will CANCEL the call and return GCEV_DISCONNECTED event sent to application with reason IPEC_InternalReasonSRTPCryptoMismatch.

Receiving calls

When the SRTP feature is enabled, the GlobalCall library will generate a new Master key for each call. When a new incoming call is received containing crypto information in the SDP, GlobalCall will respond with its crypto information in the SDP answer provided in the SIP response. If a call is received that does not include crypto information, GlobalCall will not include crypto in its response. The resulting call will not utilize Secure RTP.

The table below shows documents HMP behavior during inbound call negotiation.

Local	Remote sends	Results
SRTP Disabled	Invite without SRTP	Invoke regular RTP call handling
SRTP Enabled	Invite with SRTP	Invoke Secure RTP call handling
SRTP Disabled	Invite with SRTP	Send 488 “Not Acceptable Here” response to remote and return GCEV_DISCONNECTED event sent to application.
SRTP Enabled	Invite without SRTP	Invoke regular RTP call handling (no SRTP)

Feature notes

- While HMP IP Media Library implementation supports the ability to specify multiple encryption keys, 1PCC Secure RTP feature utilizes a single key per call. Multiple key rotation is not supported.
- For more information related to Secure RTP, see Chapter 21 of the "Dialogic® IP Media Library API Programming Guide and Library Reference" https://www.dialogic.com/-/media/manuals/docs/ip_media_api_hmp_v16.pdf
- For more information related to SIP TLS, see Chapter 4, Section 29 of the "Dialogic® Global Call IP Technology Guide" https://www.dialogic.com/-/media/manuals/docs/globalcall_for_ip_hmp_v12.pdf

Support for MKV and WebM file formats

The mm_Play() and mm_Record() functions have been updated to support both audio and video tracks in a single MKV and WebM file.

For recording, the EMM_MEDIA_TYPE_AUDVID media item has been added to indicate recording both audio and video tracks to a single file. The MM_MEDIA_AUDVID structure which includes fields to specify both audio and video information has also been added. The INIT_MM_MEDIA_AUDVID() function must be used to initialize the structure.

For playback, the audio and video tracks are specified separately with both pointing to the same file.

Example code

…

```
MM_PLAY_RECORD_LIST sMmRecordList[1];
MM_RUNTIME_CONTROL sRuntimeControl;
INIT_MM_RUNTIME_CONTROL(&sRuntimeControl);

MM_MEDIA_ITEM_LIST sAudioVideoMediaItemList[1];
INIT_MM_MEDIA_ITEM_LIST(&sAudioVideoMediaItemList[0]);
INIT_MM_MEDIA_AUDVID(&(sAudioVideoMediaItemList[0].item.audvid));

sAudioVideoMediaItemList[0].ItemChain = EMM_ITEM_EOT;
sAudioVideoMediaItemList[0].item.audvid.eFileFormat = EMM_FILE_FORMAT_WEBM;
sAudioVideoMediaItemList[0].item.audvid.unOffset = 0;
sAudioVideoMediaItemList[0].item.audvid.unMode = MM_MODE_NOIFRMBEEPINITIATED;
sAudioVideoMediaItemList[0].item.audvid.szFileName = "conf_party_01.webm";
sAudioVideoMediaItemList[0].item.audvid.unAccessMode = MM_MEDIA_ACCESS_MODE_FILE;

sAudioVideoMediaItemList[0].item.audvid.vidCodec.Coding = EMM_VIDEO_CODING_VP8;
sAudioVideoMediaItemList[0].item.audvid.vidCodec.Profile = VIDEO_PROFILE_DEFAULT;
sAudioVideoMediaItemList[0].item.audvid.vidCodec.Level = VIDEO_LEVEL_DEFAULT;
sAudioVideoMediaItemList[0].item.audvid.vidCodec.ImageWidth = (eMM_VIDEO_IMAGE_WIDTH)
1280;
sAudioVideoMediaItemList[0].item.audvid.vidCodec.ImageHeight =
(eMM_VIDEO_IMAGE_HEIGHT) 720;
sAudioVideoMediaItemList[0].item.audvid.vidCodec.BitRate = (eMM_VIDEO_BITRATE)
2000000;
sAudioVideoMediaItemList[0].item.audvid.vidCodec.FramesPerSec = VIDEO_FRAMESPERSEC_15;
sAudioVideoMediaItemList[0].item.audvid.audCodec.unCoding = MM_DATA_FORMAT_OPUS;
sAudioVideoMediaItemList[0].item.audvid.audCodec.unSampleRate = MM_DRT_16KHZ;
sAudioVideoMediaItemList[0].item.audvid.audCodec.unBitsPerSample = 16;

sMmRecordList[0].ItemType = EMM_MEDIA_TYPE_AUDVID;
sMmRecordList[0].list = &(sAudioVideoMediaItemList[0]);
```

```
sMmRecordList[0].ItemChain = EMM_ITEM_EOT;
sMmRecordList[0].next = NULL;
```