Dialogic® Multimedia Software for AdvancedTCA Release 1.0

Release Update

January 23, 2008

05-2548-006

Publication Date: January 23, 2008

Document Number: 05-2548-006

About This Publication

This section contains information about the following topics:

- Purpose
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This Release Update addresses issues associated with Dialogic® Multimedia Software for AdvancedTCA Release 1.0. In addition to summarizing issues that were known as of this release, the Release Update will continue to be updated to serve as the primary mechanism for communicating new issues that arise after the release date.

Intended Audience

This Release Update is intended for all users of the Dialogic® Multimedia Platform for AdvancedTCA.

How to Use This Publication

This Release Update is organized into three sections (click the section name to jump to the corresponding section):

- Document Revision History: This section summarizes the ongoing changes and additions that are made to this Release Update after its original release. This section is organized by document revision and document section.
- Post-Release Developments: This section describes significant changes to the release subsequent to the general availability release date. For example, new features provided in service updates are described here.
- Release Issues: This section lists issues that may affect the system release hardware and software.
- Documentation Updates: This section contains corrections and other changes that apply to the documentation set that were not made to the documents prior to the release. The updates are organized by documentation category and by individual document.

Related Information

See the following for additional information:

- *http://www.dialogic.com/manuals/* (for Dialogic® product documentation)
- *http://www.dialogic.com/support/* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

Document Revision History

This Revision History summarizes the changes made in each published version of the Release Update for the Dialogic® Multimedia Platform for AdvancedTCA, which is a document that is periodically updated throughout the lifetime of the release.

Document Rev 06 – published January 23, 2008

Made global changes to reflect Dialogic Brand and changed title to "Dialogic® Multimedia Software for AdvancedTCA Release 1.0 Release Update."

In the Documentation Updates chapter:

- Removed documentation updates in Dialogic® Voice API Library Reference as a new version of document (05-2333-004) is available.
- Removed documentation updates in Dialogic® Voice API Programming Guide as a new version of document (05-2332-004) is available.
- Removed documentation updates in MSML Media Server User's Guide as a new version of document (05-2513-001) is available.

Document Rev 05 – published June 22, 2007

Updated for Service Update 81.

In the Post-Release Developments chapter:

- Added Summary of Features.
- Added Support for MSML Conferencing.
- Added Support for AMR and EVRC in MSML.

Document Rev 04 – published June 8, 2007

Updated for Service Update 80.

In the Post-Release Developments chapter:

- Added Support for QCELP Codec.

In the Release Issues chapter:

- Updated description of Known Defects: IPY00038148, IPY00038149.
- Added the following Known Defect: IPY00037689.

In the Documentation Updates chapter:

- For Dialogic® IP Media Library API Library Reference, indicated that IPM_AUDIO_CODER_INFO structure and IPM_AUDIO_CODER_OPTIONS_INFO structure have updates for QCELP; updated table of supported audio coders in IPM_AUDIO_CODER_INFO section (EVRC change); updated IPM_AUDIO_CODER_OPTIONS_INFO structure (removed CODER_OPT_EVRC_SIGNALLING_OFF, CODER_OPT_EVRC_SIGNALLING_DIM_AND_BURST_HALF_RATE, CODER_OPT_EVRC_SIGNALLING_BLANK; added CODER_OPT_SIGNALING_OFF); updated example code for IPM_AUDIO_CODER_OPTIONS_INFO structure.

Document Rev 03 – published May 1, 2007

Updated for Service Update 77.

In the new Post-Release Developments chapter:

- Added Service Update for Dialogic® Multimedia Software for AdvancedTCA Release 1.0 section.

  Added important information on the need to uninstall existing software **before** installing the new Service Update.

In the Release Issues chapter:

- Updated description of Known Defect: IPY00034680.
- Added the following Known Defects: IPY00033349 (previously Resolved), IPY00033354 (previously Resolved), IPY00038147, IPY00038148, IPY00038149.
- Added the following Resolved Defects: IPY00035425, IPY00035828, IPY00036058, IPY00036313, IPY00036386, IPY00036831, IPY00036858, IPY00036983.
- Removed the following Known (Permanent) Defect: IPY00032616 (not a defect).
- Removed the following Known Defects (duplicate entries; previously resolved in Service Update 70): IPY00036311, IPY00036314.

In the Documentation Updates chapter:

- Added a new version of the Dialogic® Multimedia Blade for ATCA Technical Product Specification to the bookshelf (previously added in Service Update 70).

Document Rev 02 – published March 30, 2007

Updated for Service Update 70.

In the Release Issues chapter:

- Added SU No. to the Issues table.
- Revised the description for Known Defect IPY00032623.
- Added the following Known Defects: IPY00036858 and unnumbered defect for MSML regarding http scheme.
- Added the following Known (Permanent) Defect: IPY00032616 (previously a Known Defect).

- Added the following Resolved Defects: IPY00036309, IPY00036311, IPY00036314, IPY00036601, IPY00036911.

In the Documentation Updates chapter:

- Added update to the Dialogic® Multimedia API Library Reference for MM_VIDEO_CODEC data structure.

Document Rev 01 – published January 2007

Initial version of document.

Post-Release Developments 1

This section describes significant changes to Dialogic® Multimedia Software for AdvancedTCA Release 1.0 subsequent to the general availability release date.

1.1 Service Update for Dialogic® Multimedia Software for AdvancedTCA Release 1.0

A Service Update for Dialogic® Multimedia Software for AdvancedTCA Release 1.0 is available. Service Updates provide fixes to known problems and may also introduce new functionality. New versions of the Service Update will be released periodically. This Release Update documents the features in the Service Updates.

For the latest service update, go to http://www.dialogic.com/support/helpweb/mmpatca/.

Caution: Depending on whether you already have a version of the Dialogic® Multimedia Software for AdvancedTCA Release 1.0 software on your system, you may need to uninstall the existing version before installing the Service Update:

- If you don’t have an existing version of the software on your system, you can install the Service Update immediately.
- If you have an existing version of the software on your system, you must **uninstall the existing version before installing the Service Update**. This Service Update includes global software changes to reflect the new Dialogic brand.

To uninstall the existing version of the software, run the uninstall script, *dlguninstall.sh*, distributed with the existing version of the software, located in /usr/dialogic/bin. The uninstall script gives you an opportunity to save your configuration settings to a backup location before starting the uninstall. After you have saved your configuration settings, run the uninstall script again. After the uninstall has completed, it is not necessary to reboot the system. However, you must log out and log in to restore the environment settings.

After successfully uninstalling the existing version of the software, you can install the new software. Instructions for installing the software are provided in the Technical Product Specification.

1.2 Summary of Features

The following table lists the features introduced in service updates after the initial release of Dialogic® Multimedia Software for AdvancedTCA Release 1.0 and where to find more information about these features.

Feature	Service Update	For more information
MSML conferencing	81	see Section 1.3, "Support for MSML Conferencing", on page 9 and *MSML Media Server User's Guide* .
AMR and EVRC in MSML	81	see Section 1.4, "Support for AMR and EVRC in MSML", on page 9.
QCELP codec	80	see Section 1.5, "Support for QCELP Codec", on page 9.

1.3 Support for MSML Conferencing

With this Service Update, the MSML conferencing feature is supported. For more information, see the *MSML Media Server User's Guide.*

1.4 Support for AMR and EVRC in MSML

With this Service Update, the AMR and EVRC audio codecs are supported in MSML. For more information about MSML, see the *MSML Media Server User's Guide.*

1.5 Support for QCELP Codec

With this Service Update, the QCELP 8 kbps and 13 kbps coder type are now supported for media streaming operations.

Updates to IPM_AUDIO_CODER_INFO

The IPM_AUDIO_CODER_INFO data structure has the following updates:

eCoderType
For QCELP:
- CODER_TYPE_QCELP_8K – QCELP, 8 kbps
- CODER_TYPE_QCELP_13K – QCELP, 13 kbps

The QCELP coder type supports frame size of 20; frames per packet of 1, 2, or 3; and eVadEnableValue of CODER_VAD_DISABLE.

Updates to IPM_AUDIO_CODER_OPTIONS_INFO

The IPM_AUDIO_CODER_OPTIONS_INFO data structure has the following additional values:

unCoderOptions
: For QCELP, set to CODER_OPT_SIGNALING_OFF.

unParm1
: For QCELP, where the media type is MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO, valid values are:

  - CODER_OPT_CDMA_RATE_REDUC - Rate reduction. Bit rate is varied to achieve a variety of average bit rates for more flexibility in bandwidth usage.

nValue1
: For QCELP 8 kbps, where unParm1=CODER_OPT_CDMA_RATE_REDUC, possible values are 0 and 4. Default value is 0.

  For QCELP 13 kbps, where unParm1=CODER_OPT_CDMA_RATE_REDUC, possible values are 0, 1, 2, 3, and 4. Default value is 0.

unParm2
: For QCELP, set to 0.

nValue2
: For QCELP, set to 0.

Release Issues 2

The table below lists issues that can affect the hardware and software supported in the Dialogic® Multimedia Platform for AdvancedTCA. The following information is provided for each issue:

Issue Type
: This classifies the type of release issue based on its effect on users and its disposition:

  - Known – A minor hardware or software issue. This category includes interoperability issues and compatibility issues. Known issues are still open but may or may not be fixed in the future.
  - Known (permanent) – A known hardware or software issue or limitation that will not be fixed in the future.
  - Resolved – A hardware or software issue that was resolved (usually either fixed or documented) in this release.

Defect No.
: A unique identification number that is used to track each issue reported via a formal Change Control System. Additional information on defects may be available via the Defect Query tool at *http://membersresource.dialogic.com/search/defects* (Note that when you select this link, you will be asked to either LOGIN or JOIN.).

SU No.
: For defects that were resolved in a Service Update, indicates the Service Update number. For defects that were resolved when the base release was generally available (before any Service Updates), a "--" is shown. For non-resolved issues, this information is left blank.

Product or Component
: The product or component to which the problem relates, for example, an API.

Description
: A summary description of the issue. For non-resolved issues, a workaround is included when available.

Issues Sorted by Type, Dialogic® Multimedia Platform

Issue Type	Defect No.	SU No.	Product or Component	Description
Known	IPY00035453		call control	User cannot select TCP or UDP as mode of transport for SIP on per call basis. It can be set for all messages during initialization.
Known	IPY00034680		CLI	When accessing the CLI, the three built-in accounts (acctmgr, admin, and craft) are not working. After the password is entered, the login prompt is returned instead of the cli prompt. ***Workaround:*** (1) Use root/public as username/password. This account will grant access to the CLI. (2) Create a new username/password combination and renew this combination every 90 days so it does not expire.

Known	IPY00034558		CLI	The man page for the adduser command has incorrect usernames and values in the EXAMPLE portion of the page.
				The correct EXAMPLE text is:
				`EXAMPLE:`
				`No 1:`
				`adduser user1 pass1`
				
				`A new User and his details can be added to the Security table. Username and Password are position defined and their values have to be entered with option.`
				
				`No 2:`
				`adduser user1 pass123 -a 50 -g 50 -i ADVENTMAINT`

Known	IPY00034557		CLI	The man page for the edituser command shows the wrong command in the EXAMPLE portion of the page - it shows adduser instead of edituser.
				The correct EXAMPLE text is:
				`EXAMPLE:`
				`No 1:`
				`edituser -p user1 pass1`
				
				`Edit and modify the details of any user in the User table. Username is position defined and their values have to be entered with option.`
				
				`NO 2:`
				`edituser -p pass123 -g 50 -a 50 -i ADVENTMAINT user1`

Issues Sorted by Type, Dialogic® Multimedia Platform (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Known	IPY00034601		clocking	The TDM clock manager does not switch to a valid clock that is located earlier in the fallback list before the failed entry. For example, if the last valid trunk clock source fails, the firmware chooses the internal oscillator as the clock source instead of a previously recovered trunk that is in the fallback list. The correct behavior is that the fallback list should be rescanned from the beginning before choosing internal oscillator. **Note:** if any clock recovers while internal oscillator is selected, then the firmware rescans the list from the beginning. ***Workaround:*** The user can reset the fallback list manually. At a minimum, there will always be a valid tx PSTN clock provided by internal oscillator.
Known	IPY00033354		conferencing (CNF)	The **cnf_GetPartyList( )** function does not return correctly when max parties (240) are entered into the conference.
Known	IPY00033349		conferencing (CNF)	CNFEV_SET_ATTRIBUTE_FAIL is received when attempting to disable tone clamping.
Known	IPY00035617		configuration	The RFC2833 payload type is currently set to 101. Even though the user may specify another payload type for RFC2833, the software transmits RFC2833 with payload type of 101.
Known	IPY00035616		configuration	The RTCP port number is not configurable by the user. RTCP port will always be RTP+1. Currently this cannot be changed by the user.
Known	IPY00035815		firmware	The firmware tries to load the Rear Transition Module (RTM) PCI driver even when the RTM is not present. The firmware displays a KILLTASK message in the /var/log/messages file because the RTM driver failed to initialize. Since the RTM is not present, the firmware should not try to initialize the RTM driver and should not treat the failure as an error. The error can be ignored.
Known	IPY00032953		firmware	QERROR_WARNING received during download with Cause Tag: 80011, Location Tag: 49111. The warning can be ignored.
Known	IPY00037689		hardware	If the Dialogic® Multimedia Blade is shut down with an RTM present, the baseboard hotswap (blue) LED remains in a blinking state and never goes to a steady state as expected to indicate shutdown is complete. **Workaround:** Wait for all software services to shut down before extracting the board.

Issues Sorted by Type, Dialogic® Multimedia Platform (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Known	IPY00038149		hardware	Shut down is not supported by opening of the Dialogic® Multimedia Blade latch. The blue LED (hotswap LED) on the blade will blink indefinitely but never turn solid to indicate shut down complete. **Workaround:** To shut down the blade, log on locally through the serial port or connect over the network and initiate a shutdown. The blade can be initialized by removing and reseating in the chassis.
Known	IPY00038148		hardware	Activation and de-activation via the shelf manager is not supported on the Dialogic® Multimedia Blade latch. The blue LED (hotswap LED) on the blade will blink indefinitely but never turn solid to indicate shut down complete. **Workaround:** To shut down the blade, log on locally through the serial port or connect over the network and initiate a shutdown. The blade can be initialized by removing and reseating in the chassis.
Known	IPY00038147		hardware	If the Dialogic® Multimedia Blade is deactivated using the shelf manager, the kernel panics while shutting down the system services and does not complete the hot swap extraction cycle. The craftsperson can then remove and/or restart the blade.
Known	IPY00036028		hardware	The PSTN Rear Transition Module (RTM) generates major events to the shelf manager stating that the sensor “RTM +1.0 V” went over the upper critical limit.
Known	IPY00036026		hardware	The Dialogic® Multimedia Blade generates major events to the shelf manager stating that the sensor “SBC +1.1V SUS” went over the upper critical limit.
Known	IPY00032623		MSML	‘audiosamplerate’ and ‘audiosamplesize’ must be specified by the MSML application server when playing a .WAV file from an offset.
Known	IPY00034143		PSTN call control	When using all 496 time slots on 16 E1 trunks, there is a 1% failure rate with DTMF digit detection.
Known (permanent)	IPY00034168		CLI	On a CLI shutdown (by dlstop), the CLI session port is not immediately released. If a user does a dlstop followed by a dlstart too quickly (within 20 seconds), the dlstart may not be able to start the CLI session. ***Workaround:*** If the CLI agent start part of dlstart fails, then the workaround is to do the following from a terminal window after around 20 seconds: `dlservices cli start` If it doesn't work immediately, try again in a few seconds. This will start the CLI agent and user will be able to use telnet again.

Issues Sorted by Type, Dialogic® Multimedia Platform (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Known (permanent)	IPY00033953		IPML	Calling **ipm_SetParm( )** without calling **ipm_Open( )** causes the library to crash and the application to exception. ***Workaround:*** Do not call any IPML functions without first opening the device.
Known (permanent)	IPY00033884		IPML	Calling **ipm_SetParm( )** asynchronously three times per device sequentially for 360 devices hangs the library. The parameters being used in **ipm_SetParm( )** are DTMFXFERMODE_RFC2833, PARMCH_RFC2833EVT_TX_PLT, and PARMCH_RFC2833EVT_RX_PLT. ***Workaround:*** The application should wait for IPMEV_SETPARM before issuing the next **ipm_SetParm( )**.
Resolved	IPY00033685	--	CLI, SNMP	Using SNMP or CLI to start the media results in the media devices not being initialized. The command will return and the download will take place, but no devices are being initialized.
Resolved		--	EVRC	Use of EVRC has to be specifically enabled.
Resolved	IPY00036983	77	hardware	The SDR file for IP only RTM (MMBRTMIP01AQ_100.SDR) cannot be downloaded using the sbcupdate utility.
Resolved	IPY00036386	77	IPML	DTMF transfer mode should be set prior to calling **dev_ReserveResourceEx( )**, otherwise it will not take effect.
Resolved	IPY00036314	70	IPML	Setting the AMR-NB codec option to BWE (Bandwidth Efficient Mode) results in a fatal error. Do not use the BWE mode for the AMR-NB codec.
Resolved	IPY00036311	70	IPML	The EVRC codec does not support 3FPP. The **ipm_StartMedia( )** call fails.
Resolved	IPY00034204	--	IPML	AMR narrow band coder has some CMR options that are not yet supported in the firmware and host libraries. Restrictions are: • The transmit bit rate will follow the remote CMR request. • The CMR in the outgoing packet will be set to default value of 15 (don't care - will accept any rate from remote side).
Resolved	IPY00034016	--	IPML	The **ipm_Listen( )** function fails when the CT Bus PCM encoding type does not match the encoding type of the first trunk (or framer) when PSTN is enabled.
Resolved	IPY00033771	--	IPML	QoS: The application failed to receive RTP and RTCP time-out event. RTCP events are not working at this time.
Resolved	IPY00033720	--	IPML	Pulling an Ethernet LAN cable from the RTM does not generate a change in line status event. RTCP events are not working at this time.

Issues Sorted by Type, Dialogic® Multimedia Platform (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00036911	70	MSML	Failure occurs when running 120 G.711 and 120 G.729 coders on the Dialogic® Multimedia Blade.
Resolved	IPY00036601	70	MSML	CMR bits in octet aligned mode are in the wrong location of the AMR-NB RTP payload.
Resolved	IPY00036309	70	MSML	Density for LBR codecs on Mindspeed is less than what it should be when mixed with PSTN channels or G.711 20ms, 30ms, 40ms channels on the Mindspeed device.
Resolved		77	MSML	An MSML media_server segmentation fault can occur using the "http://" scheme to play/record files on multiple channels at the same time.
Resolved	IPY00036058	77	MSML	User may have to re-initialize the Dialogic® Multimedia Blade if the MSML server does not exit gracefully after ^C.
Resolved	IPY00035828	77	MSML	User needs to manually set the debug state to '1' in the *RtfConfigLinux.xml* for media_svr.
Resolved	IPY00034094	--	MSML	The media section is missing from the *rtfconfiglinux.xml* file. In order to perform any media server debug tracing, the parameters need to be added and then switched on.
Resolved	IPY00033420	--	MSML	PSTN is not supported through MSML.
Resolved	IPY00033245	--	MSML	Media server does not return correct error code for record tests.
Resolved	IPY00032622	--	MSML	The play.amt shadow variable does not return the correct length of time the file has played; it returns 0.
Resolved	IPY00036831	77	PSTN call control	Turning on NLP causes channel failure.

Issues Sorted by Type, Dialogic® Multimedia Platform (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00035425	77	PSTN call control	The firmware is printing unnecessary logs to the messages file. These messages can be ignored. The messages are similar to the following: Oct 19 11:10:31 py1atcatvl16 CLIAGENT: From Component: IPMS_ShowLinkAlarm : @ LastChange:0 seconds @ Oct 19 11:13:22 py1atcatvl16 ssp_x86Linux_boot: line 3056, file src/c/tsccomp.c Oct 19 11:13:22 py1atcatvl16 ssp_x86Linux_boot: line 1198, file src/c/tsccomp.c Oct 19 11:13:22 py1atcatvl16 ssp_x86Linux_boot: line 3056, file src/c/tsccomp.c Oct 19 11:13:22 py1atcatvl16 ssp_x86Linux_boot: line 1198, file src/c/tsccomp.c Oct 19 11:13:22 py1atcatvl16 [0xffff0000]TSC.000 Received unexpected message Unknown message type(0x1244) from 0:0:2:b [0xffff0000]TSC.001 Received unexpected message Unknown message type(0x1234) from 0:0:2:b [0xffff0000]TSC.000 Received unexpected message Unknown message type(0x1244) from 0:0:2:b [0xffff0000]TSC.032 Received unexpected message Unknown message type(0x1234) from 0:0:2:b Oct 19 11:14:47 py1atcatvl16 ssp_x86Linux_boot: RtcWriteBEvtDetected error: Event type -4605 is out of range (max=127)
Resolved	IPY00034277	--	TDM bus	Receive failures in the DS1 adapter when attempting to add a clock source to the FallBackList table via SNMP. The Clock FallBack List is not functional; the user can choose a clock but will not be able to rely on a fallback list.
Resolved	IPY00034275	--	TDM bus	When no fallback list is present, the TDM clock manager in the firmware will attempt to access the non-existent list when the cable is unplugged from a PSTN trunk. This causes two identical events to be forwarded to the CLI and SNMP saying that the TDM clock has changed sources twice. This is misleading because in actuality the clock source has not changed and red alarms on a particular trunk (or trunks) should not have any effect on the default internal oscillator clock source. The Clock FallBack List is not functional; the user can choose a clock but will not be able to rely on a fallback list.
Resolved	IPY00033580	--	third party support	Using RFC2833 with coder type G723.1 63(56)K, 30ms, 1(2) fpp, VAD off, digits are missed at a rate of 1 to 2%. (An example of missing digits is received “19d6a*”, expected “159d86a*”.)

Issues Sorted by Type, Dialogic® Multimedia Platform (Continued)

Issue Type	Defect No.	SU No.	Product or Component	Description
Resolved	IPY00034276	--	video	During video record, the recorded file may have missing video RTP packets. In a 3- second recording, 1 or 2 video RTP packets are dropped about 10% of the time. The problem can occur at both low and high densities.
Resolved	IPY00036858	77	voice	On a disconnection between two R4 devices, up to 30ms of the signal that was received prior to disconnection may show up on the next connection.
Resolved	IPY00036313	77	voice	User IO in Dialogic® Voice API Library is not supported in this release.

Documentation Updates 3

The documentation updates are divided into the following sections, which correspond to the top level categories used on the online documentation navigation page:

3.1 System Release Documentation

This section contains updates to the following documents:

- Dialogic® Multimedia Software for AdvancedTCA Release 1.0 Release Guide

3.1.1 Dialogic® Multimedia Software for AdvancedTCA Release 1.0 Release Guide

There are currently no updates to this document.

3.2 Technical Product Specification

This section contains updates to the following documents:

- Dialogic® Multimedia Blade for ATCA Technical Product Specification

3.2.1 Dialogic® Multimedia Blade for ATCA Technical Product Specification

A new version of the technical product specification document (05-2535-002) has been added to the bookshelf (available in Service Update 70). See the Revision History section of this document for a description of the changes.

3.3 Programming Libraries Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Dialogic® Conferencing (CNF) API Library Reference

- Dialogic® Conferencing (CNF) API Programming Guide
- Dialogic® Device Management API Library Reference
- Dialogic® Fax Software Reference
- Dialogic® Global Call API Library Reference
- Dialogic® Global Call API Programming Guide
- Dialogic® Global Call IP Technology Guide
- Dialogic® IP Media Library API Library Reference
- Dialogic® IP Media Library API Programming Guide
- Dialogic® Multimedia API Library Reference
- Dialogic® Multimedia API Programming Guide
- Dialogic® Standard Runtime Library API Library Reference
- Dialogic® Standard Runtime Library API Programming Guide
- Dialogic® Voice API Library Reference
- Dialogic® Voice API Programming Guide

3.3.1 Dialogic® Conferencing (CNF) API Library Reference

There are currently no updates to this document.

3.3.2 Dialogic® Conferencing (CNF) API Programming Guide

There are currently no updates to this document.

3.3.3 Dialogic® Device Management API Library Reference

There are currently no updates to this document.

3.3.4 Dialogic® Fax Software Reference

There are currently no updates to this document.

3.3.5 Dialogic® Global Call API Library Reference

There are currently no updates to this document.

3.3.6 Dialogic® Global Call API Programming Guide

There are currently no updates to this document.

3.3.7 Dialogic® Global Call IP Technology Guide

Update to **Section 4.3.2, Setting Coder Information**

In Table 2, "Coders Supported for Dialogic® Host Media Processing (HMP) Software," the following note should be added:

- Global Call supports AMR-NB and EVRC coders in third-party call control (3PCC) mode only. These coder types cannot be explicitly set using the Global Call API, but can be specified using the IP Media Library (IPML) API.

See the documentation updates in Section 3.3.9, "Dialogic® IP Media Library API Programming Guide", on page 32 of this Release Update for more information on using these coders in 3PCC mode.

3.3.8 Dialogic® IP Media Library API Library Reference

Update to the **ipm_GetCTInfo( )** function

For the ATCA Multimedia Platform, a value of CT_DFHMPATCA is returned in the device family field of the CT_DEVINFO data structure.

Update to the **ipm_GetLocalMediaInfo( )** function

This function returns an invalid state error if a coder is not reserved prior to calling the **ipm_GetLocalMediaInfo( )** function. All audio/video coders must be reserved using the **dev_ReserveResourceEx( )** Dialogic® Device Management API library function prior to use of the **ipm_GetLocalMediaInfo( )** function.

Note: The restriction above does not apply when using a fax (T.38) media type. The **ipm_GetLocalMediaInfo( )** can be called successfully to retrieve fax port information without using a preceding call to **dev_ReserveResourceEx( )**.

The following information, relating to **ipm_GetLocalMediaInfo( )**, should also be included.

For the audio/video media types, only RTP information needs to be queried. RTCP information is automatically provided when querying for RTP.

Update to the **ipm_GetLocalMediaInfo( )** function

Remove the first caution of the **ipm_GetLocalMediaInfo( )** function (on page 27) and add the following text to the Description section. This clarifies eMediaType and unCount as members of data structures referenced by the **pMediaInfo** function parameter and enumerates the allowed values for the eMediaType field (in the IPM_MEDIA data structure) and corresponding unCount values.

- To retrieve RTP port information for both audio and video in a multimedia session, set the eMediaType fields to MEDIATYPE_AUDIO_LOCAL_RTP_INFO and MEDIATYPE_VIDEO_LOCAL_RTP_INFO respectively and unCount to 2. See the code example. *[Corrected below.]*
- To retrieve RTP port information for a video-only session, set the eMediaType field to MEDIATYPE_VIDEO_LOCAL_RTP_INFO and unCount to 1.
- To retrieve RTP port information for an audio-only session, set the eMediaType field to MEDIATYPE_AUDIO_LOCAL_RTP_INFO and unCount to 1.

- To retrieve T.38 fax port information, set the eMediaType field to MEDIATYPE_LOCAL_UDPTL_T38_INFO and unCount to 1.

  Note: It is not possible to retrieve T.38 fax port information together with audio and/or video port information.

  Note: The RTCP port number is the RTP port number + 1.

Update to the **ipm_GetLocalMediaInfo( )** function

In the code example of the **ipm_GetLocalMediaInfo( )** function, replace the line:

```
MediaInfo.unCount = 1;
```

with:

```
MediaInfo.unCount = 2;
```

Update to the **ipm_ReceiveDigits( )** function

Not supported for the ATCA Multimedia Platform.

Update to the **ipm_SendDigits( )** function

Not supported for the ATCA Multimedia Platform.

Updates to the **ipm_SetParm( )** and **ipm_GetParm( )** functions

In addition to the parameters documented in Table 2, “eIP_PARM Parameters and Values” on the IPM_PARM_INFO data structure reference page, the IP Media Library API supports the setting and retrieval of the following parameters relating to Packet Loss Recovery (PLR) latency.

eIP_PARM Define	Description and Values
PARMCH_LATENCYALG_AUDIO	Defines the mode of operation of the jitter buffer. By default, the jitter buffer uses “adaptive mode,” which means latency can grow until the available number of frames (PARMCH_LATENCYFRAMEMAX_AUDIO) is depleted. Using fixed mode, the number of frames buffered can grow up to 1.5 times the starting value (PARMCH_LATENCYFRAMEINIT_AUDIO). Type: Integer. Values: 0 (adaptive mode) or 1 (fixed mode). Default: 0. **Note:** Use “adaptive mode” to minimize audio loss due to abnormal packet reception conditions. Use “fixed mode” to minimize latency growth by sacrificing audio quality.

eIP_PARM Define	Description and Values
PARMCH_LATENCYFRAMEINIT_AUDIO	Specifies the amount of Packet Loss Recovery (PLR) latency (delay) that can be introduced by defining the initial number of frames that can be buffered. This parameter defines the starting value for initial latency. When "fixed mode" is enabled (see PARMCH_LATENCYALG_AUDIO), the number of frames buffered can grow up to 1.5 times the starting value. For example, if the parameter value is 6, the maximum number of frames that can be buffered is 9. The PLR module attempts to restore packets arriving at the receive end as close as possible to their original time-stamped positions. Arriving packets are decomposed into individual frames, each with a unique time stamp. Each new frame is then stored in a jitter buffer before it is sent to the decoder. This is done to allow packets arriving out of order to be inserted in the queue in the correct order. The size of this jitter buffer is defined by the number of frames stored and is controlled by the PARMCH_LATENCYFRAMEINIT_AUDIO, PARMCH_LATENCYFRAMEMAX_AUDIO, and PARMCH_LATENCYALG_AUDIO parameters. Type: Integer. Values: 1 to PARMCH_LATENCYFRAMEMAX_AUDIO (frames). Default: 6. **Note:** While the number of frames to be buffered should be set as high as possible for best quality, too high a value will add unnecessary latency to the system. Generally, the number of frames buffered should be the same size or slightly larger than the number of frames per packet.
PARMCH_LATENCYFRAMEMAX_AUDIO	Defines the maximum number of frames to be buffered in the Packet Loss Recovery (PLR) frame list. This parameter adds latency only when the buffer is already filled and additional frames arrive before there is space in the buffer. This provides for bursts of packets to arrive, which would have to be discarded otherwise. The PLR module attempts to restore packets arriving at the receive end as close as possible to their original time-stamped positions. Arriving packets are decomposed into individual frames, each with a unique time stamp. Each new frame is then stored in a jitter buffer before it is sent to the decoder. This is done to allow packets arriving out of order to be inserted in the queue in the correct order. The size of this jitter buffer is defined by the number of frames stored and is controlled by the PARMCH_LATENCYFRAMEINIT_AUDIO, PARMCH_LATENCYFRAMEMAX_AUDIO, and PARMCH_LATENCYALG_AUDIO parameters. Type: Integer. Values: 30 to 200 (frames). Default: 100. **Note:** 1. On Windows® systems, the upper limit of the value is dependent on the channel density and the kernel memory availability. 2. While the number of frames to be buffered should be set as high as possible for best quality, too high a value will add unnecessary latency to the system. Generally, the number of frames buffered should be the same size or slightly larger than the number of frames per packet.

Update to the **ipm_StartMedia( )** function

An invalid state error is returned if a coder is not reserved prior to calling the **ipm_StartMedia( )** function. All audio coders must be reserved using the **dev_ReserveResourceEx( )** Dialogic® Device Management API library function prior to use in the **ipm_StartMedia( )** function.

Update to **Chapter 3, Events**

The IPMEV_QOS_ALARM event description, on page 84, indicates that "No data is returned in the event." This statement is misleading because it implies that it is not possible to determine which alarm triggered the event. Replace the description with the following text:

IPMEV_QOS_ALARM

Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when desired QoS alarm gets triggered. For information on determining which alarm triggered the event, see the code example, and specifically the CheckEvent( ) function definition, in the *Dialogic® IP Media Library API Programming Guide* section on "Example Code for QoS Alarm Handling."

Update to the **IPM_AUDIO_CODER_INFO** data structure

Update for the eCoderType field: The following possible values have been added for specifying the transmit bit rate when using AMR coders:

- CODER_TYPE_AMRNB_4_75k – 4.75 kbit/s, CMR type 0
- CODER_TYPE_AMRNB_5_15k – 5.15 kbit/s, CMR type 1
- CODER_TYPE_AMRNB_5_9k – 5.9 kbit/s, CMR type 2
- CODER_TYPE_AMRNB_6_7k – 6.7 kbit/s, CMR type 3
- CODER_TYPE_AMRNB_7_4k – 7.4 kbit/s, CMR type 4
- CODER_TYPE_AMRNB_7_95k – 7.95 kbit/s, CMR type 5
- CODER_TYPE_AMRNB_10_2k – 10.2 kbit/s, CMR type 6
- CODER_TYPE_AMRNB_12_2k – 12.2 kbit/s, CMR type 7
- CODEC_AUDIO_AMRNB_NONE – Don't care, CMR type 15. Indicates that the software does not care what bit rate it receives, the eCoderType field sets the CMR value in the transmitted packet.

For the MEDIATYPE_AUDIO_LOCAL_CODER_INFO media type, all of the above values are valid. For the MEDIATYPE_AUDIO_REMOTE_CODER_INFO media type, all values are valid except CODEC_AUDIO_AMRNB_NONE.

Update for the eFrameSize field: The only supported value for AMR is CODER_FRAMESIZE_20. This is not a technical limitation, but part of the protocol.

Update for the unFramesPerPkt field: For AMR, the only supported value is 1 frame per packet.

Update for the eVadEnable field: For AMR, setting this field to CODER_VAD_ENABLE is required.

Update for the unCoderPayloadType field: For AMR, negotiated by call control.

Update for the unRedPayloadType field: For AMR, negotiated by call control.

Update for the supported audio coders table: Table 1, "Supported Audio Coder Properties," should include the following additional coder information:

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
CODER_TYPE_AMRNB_4_75k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_5_15k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_5_9k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_6_7k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_7_4k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_7_95k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_10_2k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_12_2k	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_NONE	fixed at 20	fixed at 1	Must be CODER_VAD_ENABLE
CODER_TYPE_EVRC	fixed at 20	1, 2, or 3	Must be CODER_VAD_DISABLE

Update to the **IPM_AUDIO_CODER_INFO** data structure

Update for the eCoderType field. The QCELP 8 kbps and 13 kbps coder type are available as of Service Update 80. For more information, see Section 1.5, "Support for QCELP Codec", on page 9.

Update to the **IPM_MEDIA** data structure

Replace the IPM_MEDIA data structure field description for eMediaType with the following corrected content:

eMediaType

type of media used to start an IP session

The eIPM_MEDIA_TYPE data type is an enumeration which defines the following values:

Audio Values:

- MEDIATYPE_AUDIO_LOCAL_CODER_INFO – local receive coder information
- MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO – options when using AMR and EVRC coders at the local side
- MEDIATYPE_AUDIO_LOCAL_RTCP_INFO – local RTCP audio port information
- MEDIATYPE_AUDIO_LOCAL_RTP_INFO – local RTP audio port information
- MEDIATYPE_AUDIO_REMOTE_CODER_INFO – remote receive coder information
- MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO – options when using AMR and EVRC coders at the remote side
- MEDIATYPE_AUDIO_REMOTE_RTCP_INFO – remote RTCP audio port information
- MEDIATYPE_AUDIO_REMOTE_RTP_INFO – remote RTP audio port information

Video Values:

- MEDIATYPE_VIDEO_LOCAL_CODER_INFO – local receive video coder information
- MEDIATYPE_VIDEO_LOCAL_RTCP_INFO – local RTCP video port information
- MEDIATYPE_VIDEO_LOCAL_RTP_INFO – local RTP video port information

- MEDIATYPE_VIDEO_REMOTE_CODER_INFO – remote receive video coder information
- MEDIATYPE_VIDEO_REMOTE_RTCP_INFO – remote RTCP video port information
- MEDIATYPE_VIDEO_REMOTE_RTP_INFO – remote RTP video port information

Fax Values:

- MEDIATYPE_FAX_SIGNAL – fax signal information to be transmitted towards IP during fax transmissions
- MEDIATYPE_LOCAL_UDPTL_T38_INFO – local UDP packet T.38 information
- MEDIATYPE_REMOTE_UDPTL_T38_INFO – remote UDP packet T.38 information

Note: The following eMediaType equates are also provided for backward compatibility (the earlier generic names are equated with audio port values). However, it is recommended that the AUDIO values be used.

#define MEDIATYPE_REMOTE_RTP_INFO MEDIATYPE_AUDIO_REMOTE_RTP_INFO

#define MEDIATYPE_LOCAL_RTP_INFO MEDIATYPE_AUDIO_LOCAL_RTP_INFO

#define MEDIATYPE_REMOTE_RTCP_INFO MEDIATYPE_AUDIO_REMOTE_RTCP_INFO

#define MEDIATYPE_LOCAL_RTCP_INFO MEDIATYPE_AUDIO_LOCAL_RTCP_INFO

#define MEDIATYPE_REMOTE_CODER_INFO MEDIATYPE_AUDIO_REMOTE_CODER_INFO

#define MEDIATYPE_LOCAL_CODER_INFO MEDIATYPE_AUDIO_LOCAL_CODER_INFO

The IPM_MEDIA data structure also includes a new AudionCoderOptionsInfo field (of type IPM_AUDIO_CODER_OPTIONS_INFO) in the union that provides option details. See details of the IPM_AUDIO_CODER_OPTIONS_INFO structure following in this section of the Release Update.

New function and data structure

A new function, **ipm_GetCapabilities( )**, and new data structure, IPM_AUDIO_CODER_OPTIONS_INFO, have been added to the IPML API. See below for details.

ipm_GetCapabilities( )

Name:	ipm_GetCapabilities(nDeviceHandle, a_CapType, a_num, a_CapabilitiesArray[], a_usMode);	
Inputs:	int nDeviceHandle	• IP Media device handle
	eCAPABILITY_TYPE a_CapType	• capability type to be retrieved
	unsigned int a_num	• number of entries in the capability array
	IPM_CAPABILITIES a_CapabilitiesArray[]	• capability array
	unsigned short a_usMode	• async or sync mode setting
Returns:	number of capabilities available -1 on failure	
Includes:	srllib.h ipmlib.h	
Category:	Media Session	
Mode:	synchronous	

■ Description

The **ipm_GetCapabilities( )** function returns the number of capabilities of the specified type (for example, coders) and details of each capability supported by an active HMP software license. The number of capabilities available may be greater than the number specified by the **a_num** input parameter, therefore the following rules apply:

- If **a_num** is zero and/or **a_CapabilitiesArray[]** is NULL, this function returns only the number of capabilities available; no capability detail is retrieved.
- If **a_num** is larger than the number of capabilities available (the return value), **a_CapabilitiesArray[]** is filled with details of all capabilities and the remaining allocated memory is unused.
- If **a_num** is smaller than the number of capabilities available (the return value), **a_CapabilitiesArray[]** is filled with details of **a_num** capabilities (that is, as many as will fit); details of the remaining capabilities are not retrieved.

Parameter	Description
n_DeviceHandle	handle of the IP Media device
a_CapType	capability type, for example CAPABILITY_CODERLIST
a_num	the number of entries in the capability array
a_CapabilitiesArray[]	the capability array
a_usMode	operation mode Set to EV_SYNC for synchronous execution

The datatype for the **a_CapabilitiesArray[]** parameter is a union, IPM_CAPABILITIES, which is defined as follows:

```
typedef struct ipm_capabilities_tag
{
   unsigned int version;
   union
   {
      IPM_CODER_INFO Coder;
      // Future types here.
   };
}IPM_CAPABILITIES;
```

In this union, the IPM_CODER_INFO data structure provides coder details such as coder type, frame size, number of frames per packet, VAD enable/disable information, and payload-related information.

The datatype for the **a_CapType** parameter is eCAPABILITY_TYPE, an enumeration that is defined as follows:

```
enum eCAPABILTIY_TYPE
{
   CAPABILITY_CODERLIST;
}
```

The **ipm_GetCapabilities( )** function is supported in synchronous mode only. If asynchronous mode (a_usMode = EV_ASYNC) is specified, an error is generated.

■ Cautions

None.

■ Errors

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

EIPM_BADPARM
    Invalid parameter

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_ERROR
    System error

Example

In this example, the first **ipm_GetCapabilities( )** call retrieves only the number of capabilities available (count). That number is then used to allocate the right amount of memory and retrieve details of all the capabilities.

```
#include <ipmlib.h>

// Code follows:

unsigned int count;
IPM_CODER_INFO *caps
int i;

unsigned int count=ipm_GetCapabilties(dev,CAPABILITY_CODERLIST,0,NULL,SYNC);
caps=(IPM_CAPABILITIES *malloc(sizeof(IPM_CAPABILITIES)*count);


// check for memory error here

count=ipm_GetCapabilties(dev,CAPABILITY_CODERLIST,count,caps,SYNC);

for (i=0;i<count;i++)
{
    printf("RFC 1890 Coder Type %ui supported\n",caps[i].Coder.unCoderPayloadType);
}


// Free coder list here
free(caps);
```

See Also

None.

IPM_AUDIO_CODER_OPTIONS_INFO

```
typedef struct ipm_audio_coder_options_info_tag
{
      unsigned int unVersion;
      unsigned int unCoderOptions;
      unsigned int unParm1;
      int nValue1;
      unsigned int unParm2;
      int nValue2;
} IPM_AUDIO_CODER_OPTIONS_INFO;
```

■ Description

This data structure provides additional options when using AMR, EVRC, and QCELP audio coders. For information about QCELP (available as of Service Update 80), see Section 1.5, "Support for QCELP Codec", on page 9.

Note: All unused fields in the IPM_AUDIO_CODER_OPTIONS_INFO structure must be set to 0.

■ Field Descriptions

The fields of the IPM_AUDIO_CODER_OPTIONS_INFO data structure are described as follows:

Note: For EVRC, where the media type is MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO, all field values are ignored by the firmware.

unVersion
: set to IPM_AUDIO_CODER_OPTIONS_INFO_VERSION

unCoderOptions
: For AMR coders where the media type is MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO, valid values are:

  - CODER_OPT_AMR_EFFICIENT - Minimize the amount of network bandwidth
    or
    CODER_OPT_AMR_OCTET - Make packet parsing easier for the AMR application

  For AMR coders where the media type is MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO, valid values that can be ORed together are:

  - CODER_OPT_AMR_EFFICIENT (see description above)
    or
    CODER_OPT_AMR_OCTET (see description above)
  - CODER_OPT_AMR_CMR_TRACK - Specify that the transmit bit rate should follow the CMR value in the received packet
    or
    CODER_OPT_AMR_CMR_LIMIT - Specify that the transmit bit rate should follow the CMR value in the received packet with a maximum value specified by the preferred bit rate.

  For EVRC, set to CODER_OPT_SIGNALING_OFF.

unParm1

set to a value from the eIPM_CODER_OPTION_PARMS enumeration

For AMR, set to 0.

For EVRC, where the media type is MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO, valid value is:

- CODER_OPT_INTERLEAVE_LENGTH - Reduce the listener's perception of data loss by spreading such a loss over non-consecutive vocoder frames.
- CODER_OPT_CDMA_RATE_REDUC - Rate reduction. Bit rate is varied to achieve a variety of average bit rates for more flexibility in bandwidth usage.

nValue1

The value set here is for the parameter specified in unParm1.

For AMR, set to 0.

For EVRC, where unParm1=CODER_OPT_INTERLEAVE_LENGTH, possible values are in the range 0 to 7. The default value is 0.

For EVRC, where unParm1=CODER_OPT_CDMA_RATE_REDUC, possible values are 0 and 4. Default value is 0.

unParm2

For AMR, set to 0.

For EVRC, set to 0.

nValue2

For AMR, set to 0.

For EVRC, set to 0.

Example

The following code example shows how to set options when using an EVRC coder type:

```
... /* Setup IP address here */

// Local Audio Coder

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_EVRC;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 2;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_DISABLE;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0;

unCount++;

// Remote Audio Coder

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_EVRC;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 2;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_DISABLE;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0 ;

unCount++;
```

```
ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unVersion =
IPM_AUDIO_CODER_OPTIONS_INFO_VERSION;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unCoderOptions =
CODER_OPT_SIGNALING_OFF;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unParm1 =
CODER_OPT_INTERLEAVE_LENGTH;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.nValue1 = 6;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unParm2 = 0;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.nValue2 = 0;

unCount++;

ipmMediaInfo.unCount = unCount;
```

3.3.9 Dialogic® IP Media Library API Programming Guide

Using AMR coders for narrow band audio and using enhanced variable rate codecs
The following information, which is applicable to the ATCA Multimedia Platform, should be added to the programming guide, possibly as a new chapter.

Using AMR Coders for Narrow Band Audio

Description

AMR is an adaptive multi-rate speech codec. During operation, both local and remote sides can request a change in the bit rate and dynamically adjust the bandwidth. The protocol uses the following:

- A Frame Type (FT) to indicate the transmitted bit rate
- A Codec Mode Request (CMR) value to request a particular bit rate in every packet

To control the bit rate, AMR assumes that all connections are bi-directional.

This feature is specific to AMR-NB and excludes support for AMR-WB and AMR-WB+, which support wideband audio and some other formats not addressed by AMR.

The codec supports the following bit rates:

- 12.2 kbit/s (GSM EFR)
- 10.2 kbit/s
- 7.95 kbit/s
- 7.40 kbit/s (IS-641)
- 6.70 kbit/s (PDC-EFR)
- 5.90 kbit/s
- 5.15 kbit/s
- 4.75 kbit/s
- 1.80 kbit/s (assuming SID frames are continuously transmitted)

Note: The 1.80 kbit/s rate is not actually a voice signal, but the bit rate consumed when Voice Activation Detection (VAD) is processing a silence.

This feature is only supported when using the Session Initiation Protocol (SIP). None of the available Session Description Protocol (SDP) options are currently supported through Global Call (that is, direct first-party call control). The options are only available using third-party call control (3PCC), where the application is responsible for interpreting received SDP text strings and for constructing all outbound SDP text strings.

Dialogic® IP Media Library API Support

The bit rate can be controlled using a new media type in the IPM_MEDIA structure. The Dialogic® IP Media Library API allows the application to provide a preferred bit rate and a rule to determine how changes in the received CMR value control the transmitted bit rate.

Preferred bit rate

The preferred bit rate is specified by setting the eMediaType in the IPM_MEDIA structure to MEDIATYPE_AUDIO_REMOTE_CODER_INFO and setting the CoderInfo.eCoderType to the desired transmit bit rate.

CMR value

The transmitted CMR value is specified by setting the eMediaType in the IPM_MEDIA structure to MEDIATYPE_AUDIO_LOCAL_CODER_INFO and setting the CoderInfo.eCoderType to the desired CMR value.

The CMR rules are specified by a new media type called MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO. A new field AudioCoderOptionsInfo (of type IPM_AUDIO_CODER_OPTIONS_INFO) has been added to the IPM_MEDIA union. The CMR control is implemented by allowing the host application to specify one of two rules.

- The first rule, "CMR Tracking," indicates that the transmit bit rate should follow the CMR value in the received packet.
- The second rule, "CMR Limit," indicates that the transmit bit rate should follow the CMR value in the received packet with a maximum value specified by the preferred bit rate.

With both CMR rules, it is necessary to specify a preferred rate to avoid the specific case of determining what rate to transmit at before the first CMR value is received. The software will transmit at the preferred rate before the first packet is received or when a CMR value of 15 (don't care) is received from the opposite side.

The CMR rules are set in the AudioCoderOptionsInfo.unCoderOptions field by ORing in either CODER_OPT_AMR_CMR_TRACK or CODER_OPT_AMR_CMR_LIMIT.

Specifying a CMR rule is mandatory and the rules are mutually exclusive.

Note: The CMR rules are not used by the MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO eMediaTypes.

Supported RTP Payload Format

AMR supports two different formats for the RTP payload:

- "Bandwidth efficient," to minimize the amount of network bandwidth.

- "Octet-aligned," to make the packet parsing easier for the AMR application.

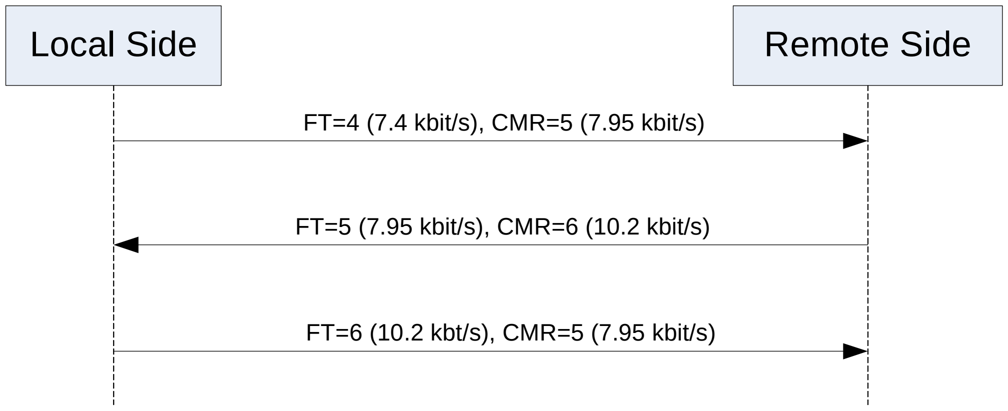
The AMR RTP payload format is specified in the new IPM_AUDIO_CODER_OPTIONS_INFO AudioCoderOptionsInfo structure (in the IPM_MEDIA structure). The MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO media type controls the receive side, and the MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO media type controls the transmit side.

The IPM_MEDIA structure contains a union with an IPM_AUDIO_CODER_OPTIONS_INFO structure. The AMR RTP payload format is controlled by ORing in either CODER_OPT_AMR_EFFICIENT or CODER_OPT_AMR_OCTET in the AudioCoderOptionsInfo.unCoderOptions field.

Note: Specifying an AMR RTP payload is mandatory and the formats are mutually exclusive.

Example 1

In this example, the host application wants to transmit at the bit rate requested by the incoming CMR value. The following diagram depicts this use case, where the local side is the host application.


Local Side
Remote Side
FT=4 (7.4 kbit/s), CMR=5 (7.95 kbit/s)
FT=5 (7.95 kbit/s), CMR=6 (10.2 kbit/s)
FT=6 (10.2 kbt/s), CMR=5 (7.95 kbit/s)

Note: While the diagram above implies an immediate reaction to a CMR from the other side, in reality, the other side's response to a CMR may take a few packets.

The sequence of activities is as follows:

1. The host is transmitting at the preferred bit rate set by the host application via the remote audio coder settings, in this case 7.4 kbit/s.

2. The host sets its CMR value to 7.95 kbit/s via the local audio coder setting.

3. The host sets its RTP payload to efficient bandwidth and CMR tracking mode.

4. The host receives its first packet from the remote side with the CMR value higher than the preferred rate.

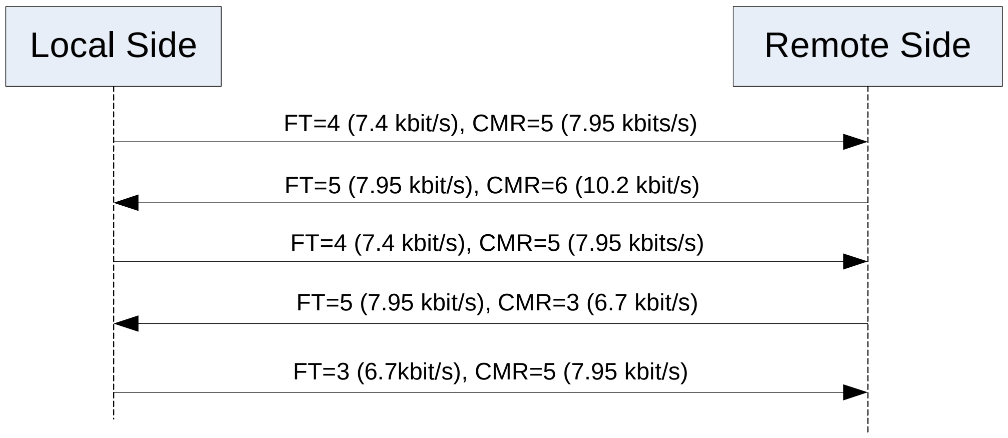
5. The host changes the transmitted bit rate to match the CMR value.

The following code demonstrates the configuration required to handle this scenario.

```
...
/* Setup IP address here */

// Local Audio Coder
ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_AMRNB_7_95k;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 1;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_ENABLE
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0
unCount++;

// Remote Audio Coder
ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_AMRNB_7_4k;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 1;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_ENABLE
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0
unCount++;

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unVersion =
                                                       IPM_AUDIO_CODER_OPTIONS_INFO_VERSION;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unCoderOptions=
                                                       CODER_OPT_AMR_EFFICIENT;
unCount++;

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unVersion =
                                                       IPM_AUDIO_CODER_OPTIONS_INFO_VERSION;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unCoderOptions=
                                              CODER_OPT_AMR_CMR_TRACK | CODER_OPT_AMR_EFFICIENT;
unCount++;
ipmMediaInfo.unCount = unCount;
...
```

Example 2

In this example, the host application wants to limit the transmitted bit rate to less than the preferred value. The following diagram shows this case, where the local side is the host application.


Local Side
Remote Side
FT=4 (7.4 kbit/s), CMR=5 (7.95 kbits/s)
FT=5 (7.95 kbit/s), CMR=6 (10.2 kbit/s)
FT=4 (7.4 kbit/s), CMR=5 (7.95 kbits/s)
FT=5 (7.95 kbit/s), CMR=3 (6.7 kbit/s)
FT=3 (6.7kbit/s), CMR=5 (7.95 kbit/s)

Note: While the diagram above implies an immediate reaction to a CMR from the other side, in reality, the other side's response to a CMR may take a few packets.

The sequence of activities is as follows:

1. The host is transmitting at the preferred bit rate set by the host application via the remote audio coder settings, in this case 7.4 kbit/s.

2. The host sets its initial CMR value to 7.95 kbit/s via the local audio coder setting.

3. The host sets its RTP payload to efficient bandwidth and CMR limit mode.

4. The host receives it first packet from the remote side with the CMR value higher than the preferred rate.

5. The host leaves the transmitted bit rate at the preferred value.

6. The host receives a packet from the remote side with a CMR value less than the preferred rate.

7. The host changes the transmitted bit rate to match the received CMR value.

The following code demonstrates the configuration required to handle this scenario.

```
...
/* Setup IP address here */

// Local Audio Coder
ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType =
CODER_TYPE_AMRNB_7_95k;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 1;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_ENABLE
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0
unCount++;

// Remote Audio Coder
ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_AMRNB_7_4k;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 1;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_ENABLE
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0
unCount++;

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo = {0};
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unVersion =
                                                IPM_AUDIO_CODER_OPTIONS_INFO_VERSION;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unCoderOptions=
                                              CODER_OPT_AMR_CMR_LIMIT | CODER_OPT_AMR_EFFICIENT;
unCount++;

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo = {0};
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unVersion =
                                                IPM_AUDIO_CODER_OPTIONS_INFO_VERSION;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unCoderOptions=
                                                CODER_OPT_AMR_EFFICIENT;
unCount++
ipmMediaInfo.unCount = unCount;
```

Using Enhanced Variable Rate Codecs

Enhanced Variable Rate Codecs (EVRCs) as specified in *RFC 3558 - RTP Payload Format for Enhanced Variable Rate Codecs (EVRC) and Selectable Mode Vocoders (SMV)* are supported.

When using EVRCs, more than one codec data frame may be included in a single interleaved/bundled packet by a sender. This is accomplished by:

- Bundling - A technique used to spread the transmission overhead of the RTP and payload header over multiple vocoder frames.
- Interleaving - A technique used to reduce the listener’s perception of data loss by spreading such a loss over non-consecutive vocoder frames.

EVRC and similar vocoders can compensate for an occasional lost frame, but speech quality degrades exponentially with consecutive frame loss.

Bundling is done using the Frames Per Packet field in the MEDIATYPE_AUDIO_REMOTE_CODER_INFO structure, as is the case with most codecs. Frames per packet values of 1 to 3 are allowed. Note that the RFC definition of bundle describes the number of additional frames per packet, so a frame per packet value of 1 would be the same as a bundle value of 0.

Interleaving is accomplished using parameter 1 of the MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO structure. This parameter can be set from 0 to 7; 0 indicates do not use interleaving. Interleaving should only be used when the frames per packet value is 2 or more.

When using the IP Media Library API, interleaving option can be specified in the IPM_AUDIO_CODER_OPTIONS_INFO data structure.

Resource reservation of audio coders

The following information, which is applicable to the ATCA Multimedia Platform, should be added to the programming guide, possibly as a new chapter.

Resource Reservation of Audio Coders

The following sections describe the use of the IPML API and Device Management API in various scenarios for resource reservation of audio coders.

Outbound Call

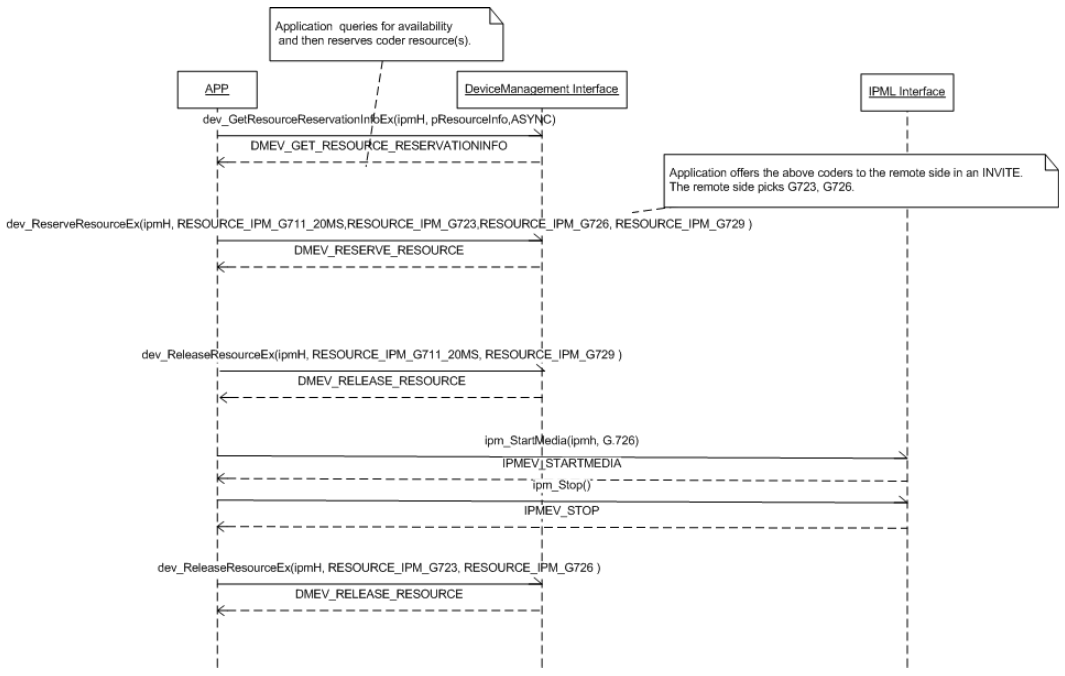
1. Application reserves coders (RESOURCE_IPM_G711_20MS, RESOURCE_IPM_G723, RESOURCE_IPM_G726, RESOURCE_IPM_G729) and offers the reserved coders to the remote side.

2. The remote side picks G723, G726.

3. Application releases (RESOURCE_IPM_G711_20MS, RESOURCE_IPM_G729). This leaves (RESOURCE_IPM_G723, RESOURCE_IPM_G726) as the reserved audio coders.

4. The application proceeds with the call.

5. After the call is done, the application releases the coders (RESOURCE_IPM_G723, RESOURCE_IPM_G726). (Alternatively, the application can release all the reserved coders by specifying the enum RESOURCE_IPM_ALL_AUDIO_CODERS.)


Application queries for availability and then reserves coder resource(s).
APP
DeviceManagement Interface
IPML Interface
dev_GetResourceReservationInfoEx(ipmH, pResourceInfo,ASYNC)
DMEV_GET_RESOURCE_RESERVATIONINFO
Application offers the above coders to the remote side in an INVITE. The remote side picks G723, G726.
dev_ReserveResourceEx(ipmH, RESOURCE_IPM_G711_20MS,RESOURCE_IPM_G723,RESOURCE_IPM_G726, RESOURCE_IPM_G729)
DMEV_RESERVE_RESOURCE
dev_ReleaseResourceEx(ipmH, RESOURCE_IPM_G711_20MS, RESOURCE_IPM_G729)
DMEV_RELEASE_RESOURCE
ipm_StartMedia(ipmh, G.726)
IPMEV_STARTMEDIA
ipm_Stop()
IPMEV_STOP
dev_ReleaseResourceEx(ipmH, RESOURCE_IPM_G723, RESOURCE_IPM_G726)
DMEV_RELEASE_RESOURCE

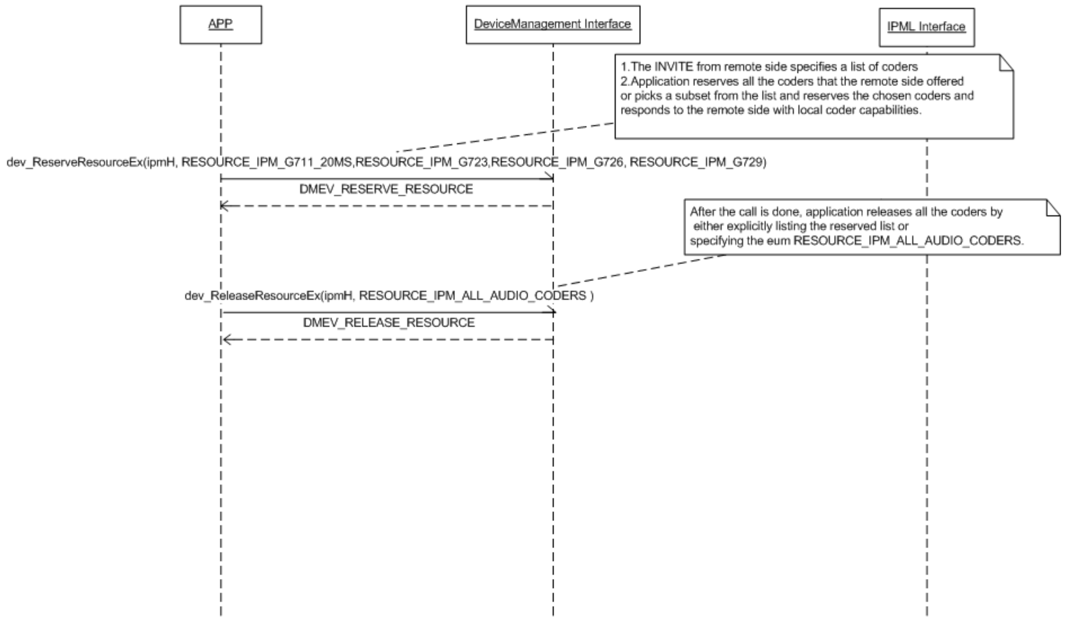
Inbound Call

1. The INVITE from remote side specifies a list of coders.

2. Application may choose to reserve all or some of the coders that the remote side offered and then respond to the remote side with local coder capabilities based on the reservation.

3. Application proceeds with the call.

4. After the call is disconnected, the application releases all the reserved coders by either explicitly listing the reserved list or specifying the enum RESOURCE_IPM_ALL_AUDIO_CODERS.


APP
DeviceManagement Interface
IPML Interface
1.The INVITE from remote side specifies a list of coders
2.Application reserves all the coders that the remote side offered or picks a subset from the list and reserves the chosen coders and responds to the remote side with local coder capabilities.
dev_ReserveResourceEx(ipmH, RESOURCE_IPM_G711_20MS,RESOURCE_IPM_G723,RESOURCE_IPM_G726, RESOURCE_IPM_G729)
DMEV_RESERVE_RESOURCE
After the call is done, application releases all the coders by either explicitly listing the reserved list or specifying the eum RESOURCE_IPM_ALL_AUDIO_CODERS.
dev_ReleaseResourceEx(ipmH, RESOURCE_IPM_ALL_AUDIO_CODERS)
DMEV_RELEASE_RESOURCE

Implicit Release by a Subsequent Successful Reserve Call

1. Application reserves (RESOURCE_IPM_G726, RESOURCE_IPM_G729).

2. Application uses RESOURCE_IPM_G726 for a media operation (ipm_StartMedia).

3. Application gets a REINVITE from the remote side requesting a change of coder.

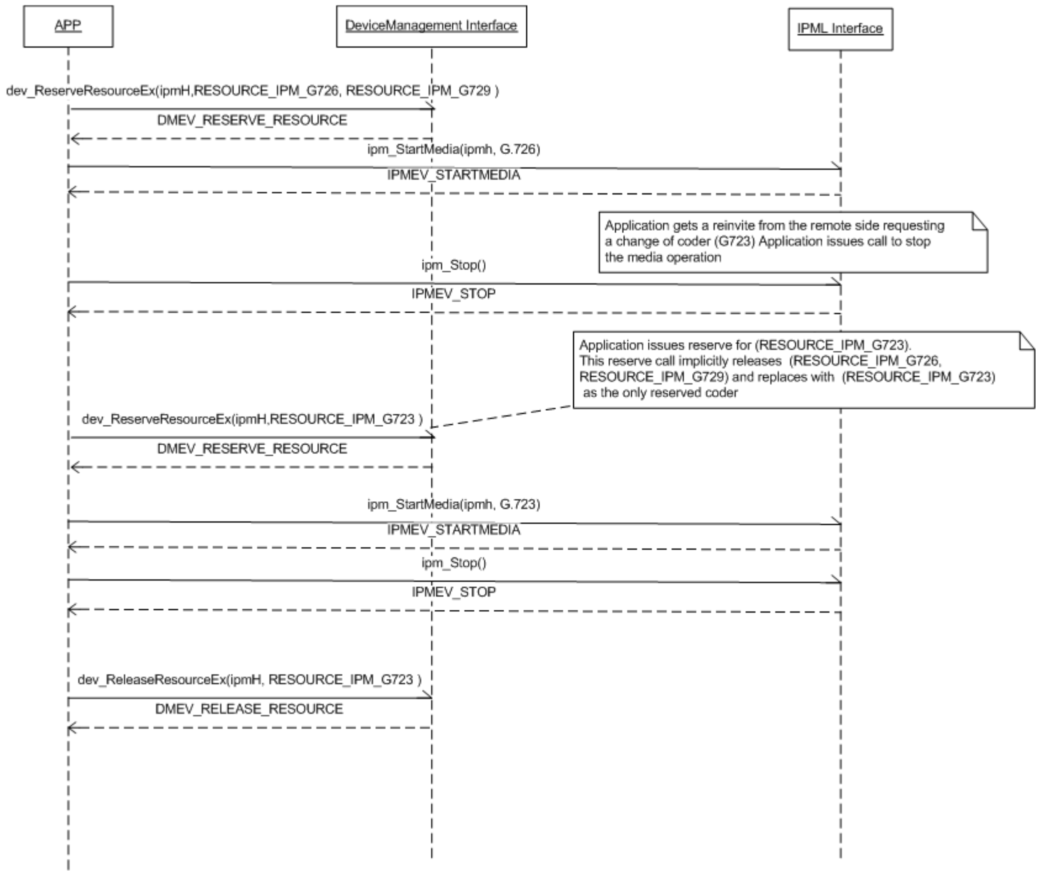
4. Application issues call to stop the media operation.

5. Application issues reserve for RESOURCE_IPM_G723. This reserve call implicitly releases (RESOURCE_IPM_G726,RESOURCE_IPM_G729) and replaces with RESOURCE_IPM_G723 as the only reserved coder.

6. Application uses RESOURCE_IPM_G723 for a media operation (ipm_StartMedia).

7. Application issues call to stop the media operation.

8. Application releases coder by issuing a call to release RESOURCE_IPM_G723.


APP
DeviceManagement Interface
IPML Interface
dev_ReserveResourceEx(ipmH,RESOURCE_IPM_G726, RESOURCE_IPM_G729)
DMEV_RESERVE_RESOURCE
ipm_StartMedia(ipmh, G.726)
IPMEV_STARTMEDIA
Application gets a reinvite from the remote side requesting a change of coder (G723) Application issues call to stop the media operation
ipm_Stop()
IPMEV_STOP
Application issues reserve for (RESOURCE_IPM_G723). This reserve call implicitly releases (RESOURCE_IPM_G726, RESOURCE_IPM_G729) and replaces with (RESOURCE_IPM_G723) as the only reserved coder
dev_ReserveResourceEx(ipmH,RESOURCE_IPM_G723)
DMEV_RESERVE_RESOURCE
ipm_StartMedia(ipmh, G.723)
IPMEV_STARTMEDIA
ipm_Stop()
IPMEV_STOP
dev_ReleaseResourceEx(ipmH, RESOURCE_IPM_G723)
DMEV_RELEASE_RESOURCE

Handling a Resource Reservation Failure

1. A call to reserve resources (RESOURCE_IPM_G726, RESOURCE_IPM_G729) fails for lack of available resources.

Note: The reserve fails when one or more of the resources requested is not available. The application can query to check on resource availability prior to issuing a reserve request. Otherwise, the application will need to query after the reserve fails to identify available/unavailable resources prior to re-issuing a reserve request.

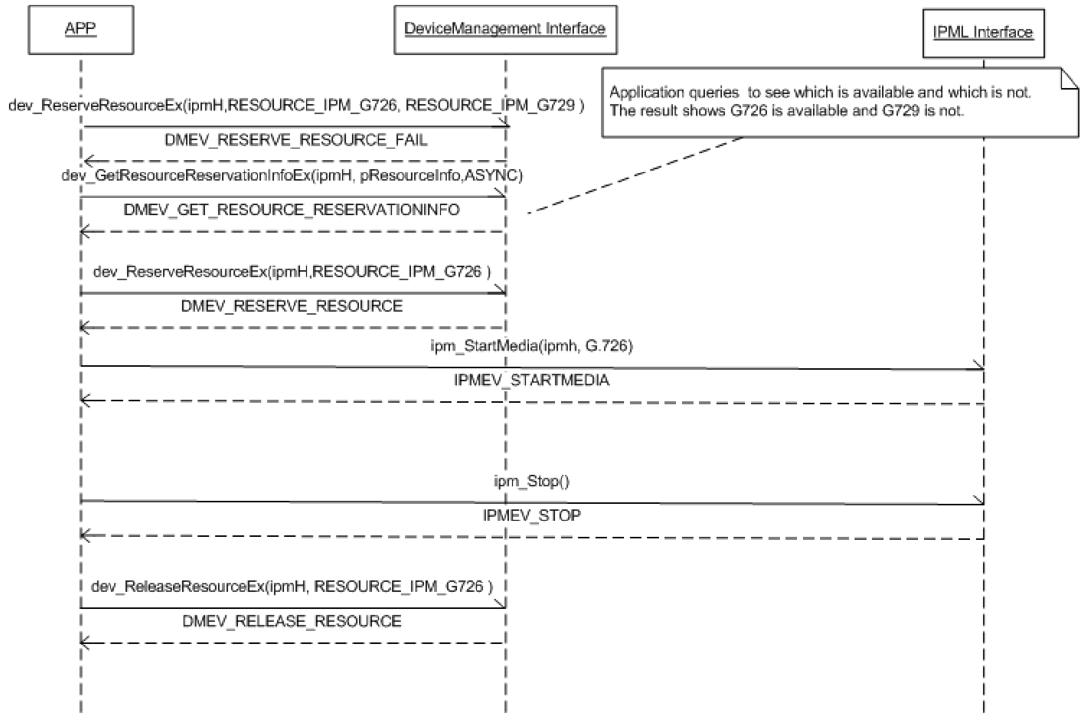
2. Application queries to check on resource availability (for this example, assume this returns RESOURCE_IPM_G726 as available).

3. Application reserves resource RESOURCE_IPM_G726.

4. Application starts media operation (ipm_StartMedia).

5. Application stops media operation (ipm_Stop).

6. Application issues call to release resource RESOURCE_IPM_G726.


APP
DeviceManagement Interface
IPML Interface
Application queries to see which is available and which is not. The result shows G726 is available and G729 is not.
dev_ReserveResourceEx(ipmH,RESOURCE_IPM_G726, RESOURCE_IPM_G729)
DMEV_RESERVE_RESOURCE_FAIL
dev_GetResourceReservationInfoEx(ipmH, pResourceInfo,ASYNC)
DMEV_GET_RESOURCE_RESERVATIONINFO
dev_ReserveResourceEx(ipmH,RESOURCE_IPM_G726)
DMEV_RESERVE_RESOURCE
ipm_StartMedia(ipmh, G.726)
IPMEV_STARTMEDIA
ipm_Stop()
IPMEV_STOP
dev_ReleaseResourceEx(ipmH, RESOURCE_IPM_G726)
DMEV_RELEASE_RESOURCE

Reservation State after a Subsequent Call to Resource Reservation Fails

1. Application issues call to reserve resources (RESOURCE_IPM_G726, RESOURCE_IPM_G729) and it succeeds.

2. Application starts media operation using G726.

3. Application receives a REINVITE request to use AMR coder.

4. Application stops media operation.

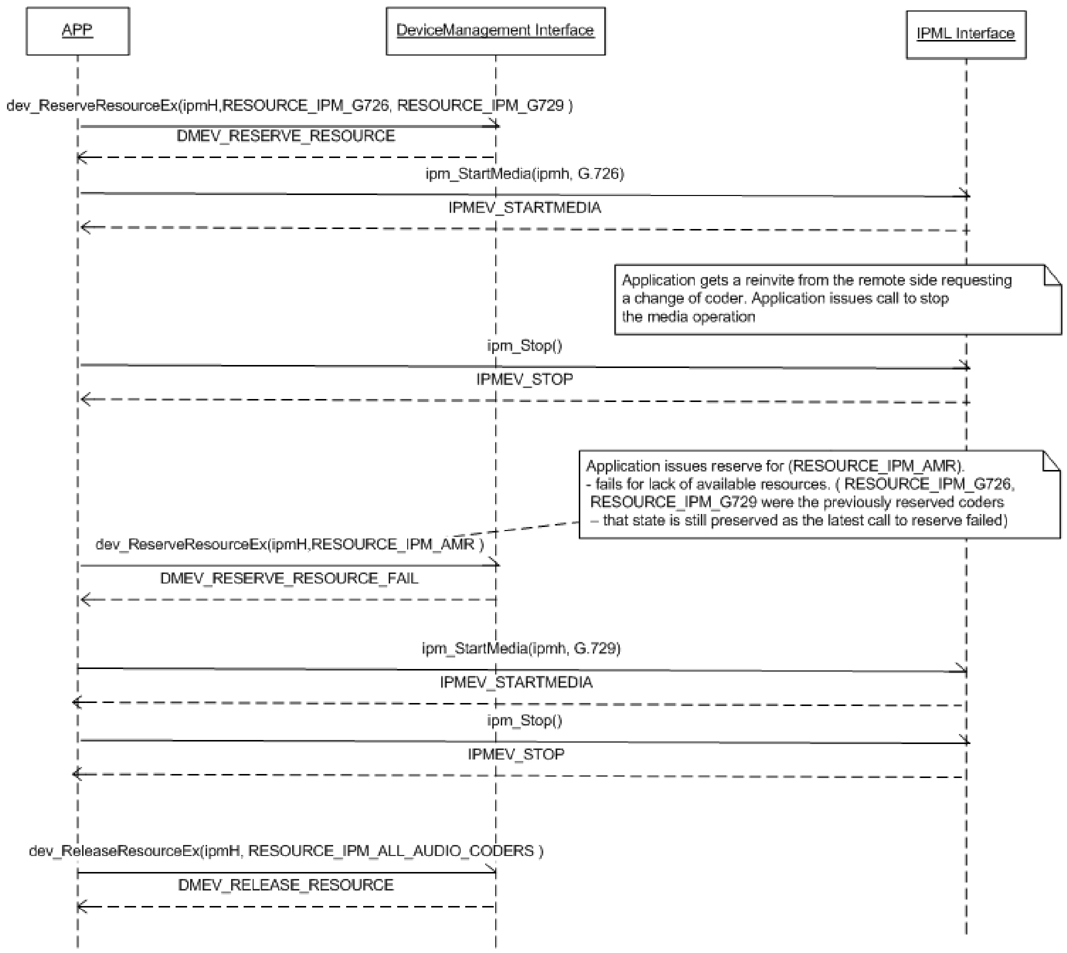
5. Application attempts to reserve RESOURCE_IPM_AMR – the call fails for lack of available resources. The previous successful reserved state is preserved. In this case, (RESOURCE_IPM_G726, RESOURCE_IPM_G729) were the previously reserved coders – that state is still preserved as the latest call to reserve failed.

6. [Application may reject the REINVITE from the remote side and continue to use any of the reserved coders at this point]

7. Start Media Operation.

8. Stop Media Operation.

9. Application issues call to release resources (RESOURCE_IPM_G726, RESOURCE_IPM_G729). It may also use a single enum RESOURCE_IPM_ALL_AUDIO_CODERS to release all the reserved audio coders as opposed to having to list every single one of them.


APP
DeviceManagement Interface
IPML Interface
dev_ReserveResourceEx(ipmH,RESOURCE_IPM_G726, RESOURCE_IPM_G729)
DMEV_RESERVE_RESOURCE
ipm_StartMedia(ipmh, G.726)
IPMEV_STARTMEDIA
Application gets a reinvite from the remote side requesting a change of coder. Application issues call to stop the media operation
ipm_Stop()
IPMEV_STOP
Application issues reserve for (RESOURCE_IPM_AMR). - fails for lack of available resources. (RESOURCE_IPM_G726, RESOURCE_IPM_G729 were the previously reserved coders – that state is still preserved as the latest call to reserve failed)
dev_ReserveResourceEx(ipmH,RESOURCE_IPM_AMR)
DMEV_RESERVE_RESOURCE_FAIL
ipm_StartMedia(ipmh, G.729)
IPMEV_STARTMEDIA
ipm_Stop()
IPMEV_STOP
dev_ReleaseResourceEx(ipmH, RESOURCE_IPM_ALL_AUDIO_CODERS)
DMEV_RELEASE_RESOURCE

3.3.10 Dialogic® Multimedia API Library Reference

Update to MM_VIDEO_CODEC data structure

For **mm_record( )**, none of the fields in this data structure may be modified, as video transcoding is not currently supported.

3.3.11 Dialogic® Multimedia API Programming Guide

There are currently no updates to this document.

3.3.12 Dialogic® Standard Runtime Library API Library Reference

There are currently no updates to this document.

3.3.13 Dialogic® Standard Runtime Library API Programming Guide

There are currently no updates to this document.

3.3.14 Dialogic® Voice API Library Reference

There are currently no updates to this document.

3.3.15 Dialogic® Voice API Programming Guide

There are currently no updates to this document.

3.4 Remote Control Interfaces Documentation

This section contains updates to the following documents:

- MSML Media Server User's Guide

3.4.1 MSML Media Server User's Guide

There are currently no updates to this document.