Dialogic®
Making Innovation Thrive™

Application Note

Video Conferencing Demo Application for Dialogic® Multimedia Products

Video Conferencing Demo Application for Dialogic® Multimedia Products

Application Note

Executive Summary

This application note presents a demo application and sample code that shows how the Dialogic® Multimedia Products can support advanced video features for multimedia applications, such as video transcoding, video transrating and image resizing, and multimedia conferencing. Topics include the design of the demo application, how conferencing and multimedia resources can be used to implement advanced video features, and how to build, configure, and run the demo.

Video Conferencing Demo Application for Dialogic® Multimedia Products

Application Note

Table of Contents

Video Conferencing Demo Application for Dialogic® Multimedia Products

Application Note

Introduction

The Dialogic® Multimedia Products support advanced video features, including video transcoding, video transrating and image resizing, and multimedia conferencing. The specific products that support these advanced features are Dialogic® Multimedia Software for AdvancedTCA Release 2.0 (MMP Software 2.0 for ATCA) for the Dialogic® Multimedia Platform for AdvancedTCA (MMP for ATCA) and Dialogic® Multimedia Kit Software Release 1.0 for PCIe (MMK Software 1.0 for PCIe), which is part of the Dialogic® Multimedia Kit for PCIe (MMK for PCIe).

The two software releases will be collectively referred to as MMP/MMK software throughout this application note.

The Dialogic® Media Toolkit API (MTK API) is also included in the software release and can be used to control the layout of video conferences and manipulate image overlays.

This application note describes a product demonstration application using advanced video features supported by MMP/MMK software to implement a video conferencing application. The VideoConfDemo application accepts incoming SIP video calls, plays a greeting video clip to each caller and then places each caller in a video conference. Dual Tone Multi-Frequency (DTMF) signaling can be used to change the conference layout and assignment of callers to conference layout regions. With each change in conference layout and region assignment, the callers are identified in the video output.

Information is included in this application note for building, configuring, and running the VideoConfDemo application. In addition, the application design is discussed, showing how the different APIs are used to implement the demo features. The demo application and sample code are available for download (see the *For More Information* section).

Video Conferencing Demo

The video conferencing demo is an application that is run using the MMP/MMK software media server and demonstrates some of the advanced video features, including video transcoding, video conferencing, and the MTK API functions.

The video conferencing demo uses the following Dialogic® APIs:

- Global Call API for 3PCC SIP IP call control
- Conferencing API for multimedia conferencing
- IP Media Library API for RTP multimedia streaming
- Device Management API for connections between devices
- Multimedia API for multimedia plays into a conference
- Media Toolkit API for video conference layout manipulation
- Media Toolkit API for video stream image overlay manipulation

MPEG-4 coder support has been introduced into MMP/MMK software, making it possible to use video transcoding to stream in one video format and convert quickly to another video format. Both MPEG-4 and H.263 video coders are supported by the video conferencing demo. For audio, G.711, G.723, G.729, and AMR coders are supported.

Video conferencing is the ability to place video-capable callers into a video conference. New video conferencing functions have been added to the Dialogic® Conferencing API to support building video conferences and setting the conference layouts. Parties attached to a conference are decoded into a common video format. The video inputs to the conference are mixed based on conference layout parameters, and the output of the conference is encoded back to the conference party. Each party receives an output stream encoded with the coder parameters specific to the party.

Video Conferencing Demo Application for Dialogic® Multimedia Products

The MTK API is a collection of libraries added to aid in video manipulation. The video conferencing demo uses the layout builder (lb_) library functions to build the video conferencing layouts and to dynamically place parties into the conferencing regions. The demo also uses the overlay builder library (ob_) functions to add an overlay that briefly displays a caller ID if there is a change in the number of callers in the conference, the assignment of callers to regions in the conference layout, or the conference layout changes.

The video conferencing application creates a video conference and waits for SIP IP video callers to call into the MMP/MMK software server to be placed into the video conference. The conference supports a configurable number of calls — up to nine parties in the conference. When the application runs, it can optionally register its extension with a SIP Registrar to allow callers to dial in by extension rather than by IP address.

The application supports SIP Video IP endpoints. 3G endpoints can be supported by using the 3G gateway demo included in the MMP/MMK software. The conferencing root frame size is CIF. Audio and video coders are negotiated with each SIP endpoint when the call is established, and transcoding, transrating, and resizing are performed to match each SIP endpoint with the conference.

When a SIP video client calls the video conferencing demo, the demo is set to answer the call and can be configured to play a greeting to the caller before placing the caller in the conference, or to place the caller directly into the conference. The conference can be configured to a default conference layout and can present up to four conference layouts to the caller. Conference layouts of 1, 4, 6, and 9 panels are supported. Once in a video conference, a SIP video caller can press "*" to cycle through the configured layouts or press "#" to rotate parties to different regions of the current conference layout.

Also, as a configurable option, the video conferencing demo can be configured to start the conference with "virtual caller" files to simulate other parties in a conference. The virtual caller files can be any pre-recorded video and audio file set, and can serve as a video greeting, marketing material, movie file, or video message clip played to party participants.

Demo Feature List

- Conference Parties:
  - Up to nine parties in a conference
- Supported coders:
  - Video coders supported: H.263, MPEG-4
  - Audio coders supported: G.711, G.723, G.729, AMR
- Conferencing layouts:
  - 1,4,6,9 party layouts supported. A subset of layouts can be configured though .configuration (.config) file.
  - DTMF control of layout:
    - "*" - Cycle through configured layouts
    - "#" - Rotate parties within frames of the current layout
- Options:
  - SIP registration
  - Optional greeting clip before being placed into conference
  - Optional "virtual callers" from multimedia clips

Building the Demo

A makefile is included in the distribution and is used to build the application using the make facility.

make options:

"make clean" — Removes intermediate object and executable files

"make" or "make release" — Builds the release version of the executable

"make debug" — Builds a debug version of the executable

Configuring the Demo

The demo may be configured by entering parameters at the command line or by using a configuration file.

The configuration parameters are:

config=xxx	use configuration file xxx, one line per item default=VideoConfDemo.cfg
port=nnn	SIP port, default=5060
maxcalls=nnn	maximum number of calls, default=4, max=9
virtualcallers=nnn	number of virtual callers, default=0, max=8
registrar=n.n.n.n	IP address of SIP Registrar, default=10.10.10.10
phonenumber=nnnn	phone number to register for, default=1000
greeting=xxxx	base filename for greeting/introduction media .pcm and .vid files, default=no greeting clip
caller1=xxxx	base filename for 1st virtual caller media .pcm and .vid files, default=caller1
caller2=xxxx	base filename for 2nd virtual caller media .pcm and .vid files, default=caller2
caller3=xxxx	base filename for 3rd virtual caller media .pcm and. vid files, default=caller3
layouts=n,n...	allowed layouts - comma delimited, by number of regions 1,4,6 and/or 9, default=1,4,6,9
initiallayout=n	initial layout by number of regions 1,4,6 or 9, default=1

A note about virtual caller media clips. One or more virtual caller media clips may be specified, and clips will be assigned to virtual callers in round-robin fashion. For example, three virtual callers are specified, caller1 is assigned "virtual1" and caller2 is assigned "virtual2". Virtual callers 1 and 3 will use the "virtual1" clip and virtual caller 2 will use the "virtual2" clip.

Running the Demo

To run the demo, execute `./VideoConfDemo`[command line parameters] at the command prompt. The application will start and initialize all the devices, displaying various progress information items in the console. Throughout execution, progress information, including API calls, events, and errors, is displayed in the console and written to the file logfile.txt.

When a SIP video call is made into the MMP for ATCA or the MMK for PCIe, the VideoConfDemo application answers the call. If the optional greeting is configured, the greeting is played to the caller before the caller is placed in the conference (see Figure 1). If the optional greeting is not configured, the caller is placed immediately into the conference. While the greeting clip is being played, the caller can end the greeting clip by pressing any DTMF digit key.

Figure 1. An Optional Greeting Being Played

The initial conference layout is set by the configuration. Callers are placed in regions of the conference layout based on the order that they called and are assigned a caller ID of "Caller1", "Caller2", etc. When a caller is added to the conference, the caller ID for all callers is displayed for five seconds (see Figure 2). The audio from a caller is mixed into the conference audio. If an audio-only call arrives for the conference, the audio is added to the conference and the video for that caller is blank. If virtual callers are configured, the virtual caller clips are played in conference layout regions not used by actual callers up to the configured number of virtual callers.

While in the conference, callers may choose to perform three operations:

1. Leave the conference by dropping the call.

2. Change the conference layout by pressing the "*" key.

3. Change the assignment of callers to conference layout regions by pressing the "#" key.

Note: When a caller operation is used, a caller ID displays for five seconds.

The conference layouts available are set by configuration. When the conference layout is changed by a caller, the conference video output is changed to the next available layout.

Callers are arranged in the conference layout regions according to the order that they called. Actual callers are followed by virtual callers. When the caller region assignments are changed by a caller, the assignments are rotated one position within the layout.

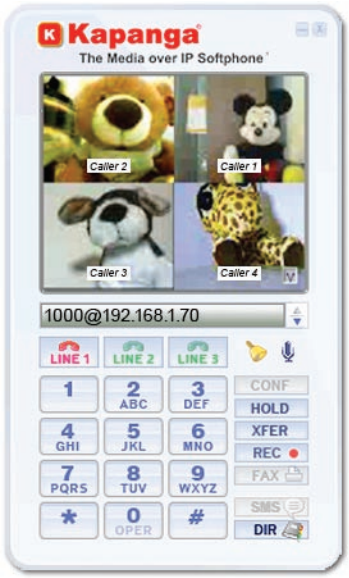

Figure 2. Four Callers in a Conference with a Caller ID Displayed for Each Caller

To exit the application, enter the letter "q" or "Q" in the console window and press return. The application will attempt to shutdown gracefully. In the event that graceful shutdown is not possible, the application will continue to try and the user will have to abort the application using Ctrl-C.

Demo Application Design

The VideoConfDemo application is designed to use an asynchronous programming model and to utilize a single thread for all event handling. The software architecture is layered such that the callflow (business) logic is layered on top of Dialogic® device handling logic. Operation of each callflow and device handling object is controlled by a state object.

The main application objects are defined by the GcManager, Conference, SipSession, VirtualCaller, and Configuration classes. The objects used to handle Dialogic devices are defined by the GcSipDevice, IpmDevice, MediaDevice, Overlay, ConferenceBoardDevice, ConferenceDevice, and ConferencePartyDevice classes.

Main Application Objects

GcManager object — Is responsible for initializing the Dialogic® Global Call API, creating the other application objects and dispatching events to the appropriate object.

Conference object — Is responsible for managing a conference device.

SipSession objects — Use the Conference object to add and remove themselves from the conference based on the callflow state, and to change the conference layout and region assignment in response to DTMF input by the caller.

SipSession objects are responsible for accepting inbound calls, playing the optional greeting clip and adding themselves to the conference. They use a SipDevice object to handle the Global Call API 3PCC SIP signaling, an IpmDevice object to handle control of the RTP media streams, a MultimediaDevice object to handle playing the optional greeting clip, a conference party device used

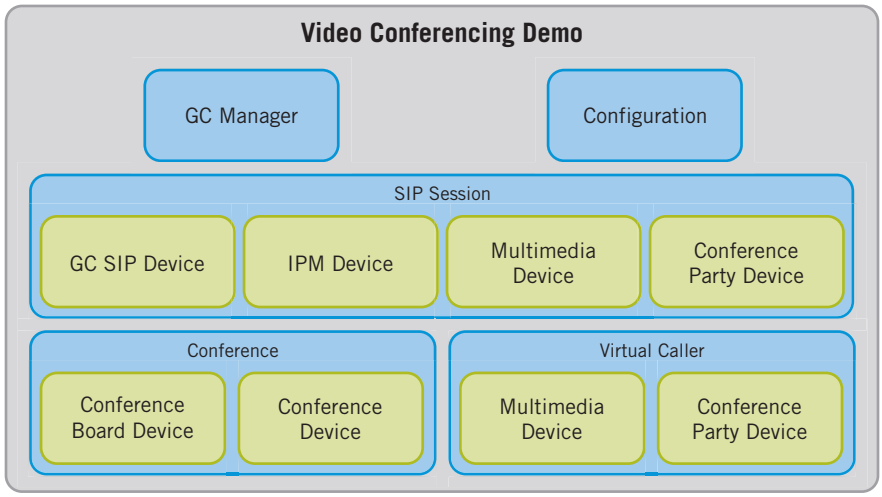

Figure 3. Software Model

to interface to the conference, and an Overlay object to display a caller identifier overlay on the video stream. The SipSession object also performs coder negotiation as part of SIP call setup. The SipSession state objects contain the logic for coordinating the SIP signaling, IpmDevice, MediaDevice, and ConferencePartyDevice objects, and for transitioning from one state to another.

VirtualCaller objects — Join the conference at startup and remain in the conference until the application ends. They use a MediaDevice object to continuously play a video clip and a ConferencePartyDevice to interface with the conference.

Objects Used to Handle the Devices

GcSipDevice objects — Are responsible for SIP call control using the Global Call API in 3PCC mode. The state machine for SIP devices has two layers to separate the device state and the call state.

IpmDevice objects — Are responsible for handling the audio and video RTP streams to SIP callers.

MediaDevice objects — Are responsible for playing media clips used by both SipSession and VirtualCaller objects.

Overlay objects — Are responsible for applying an image overlay to a video stream. SipSession objects use them to display a caller identifier when the conference video output changes layout or visible party region assignment.

A single ConferenceBoardDevice object — Is used to create ConferenceDevice and ConferencePartyDevice objects. The ConferenceDevice object is used by the Conference object to control conference attributes. ConferencePartyDevice objects are used by SipSession and VirtualCaller objects to connect to the conference.

Multimedia Conferencing

The CNF Conferencing API is used to provide conferencing features. The CNF conferencing model includes the concepts of board devices, conference devices, and conference party devices. Using the CNF API for multimedia conferencing requires only that the devices be multimedia devices. CNF devices are designated as multimedia devices by starting out with a multimedia board device. This is accomplished by using a mcxBx device name to open a multimedia conference board device. Because conference devices and conference party devices are opened using the board device, they are also multimedia devices.

The VideoConfDemo CConference and CConferenceDevice classes implement the conferencing logic. The CConference class is responsible for maintaining the lists of conference parties and virtual parties in the conference and for performing the logic necessary to change the layout and visible parties from caller DTMF input. The CConferenceDevice class is responsible for performing the actual API function calls and handling events.

Multimedia Conference Video Layout

The video layout of the multimedia conference is manipulated using Layout Builder, which is part of the MTK API.

The Layout Builder API allows for setting the size, location, and number of video regions in the conference output and for controlling which conference party appears in each. This is accomplished in two steps (see Figure 4):

1. A layout template is created. The template defines the size, location, and number of regions. Some basic properties of the regions are set in the template.

2. The layout template is then applied to the conference and visible parties are assigned to the regions.

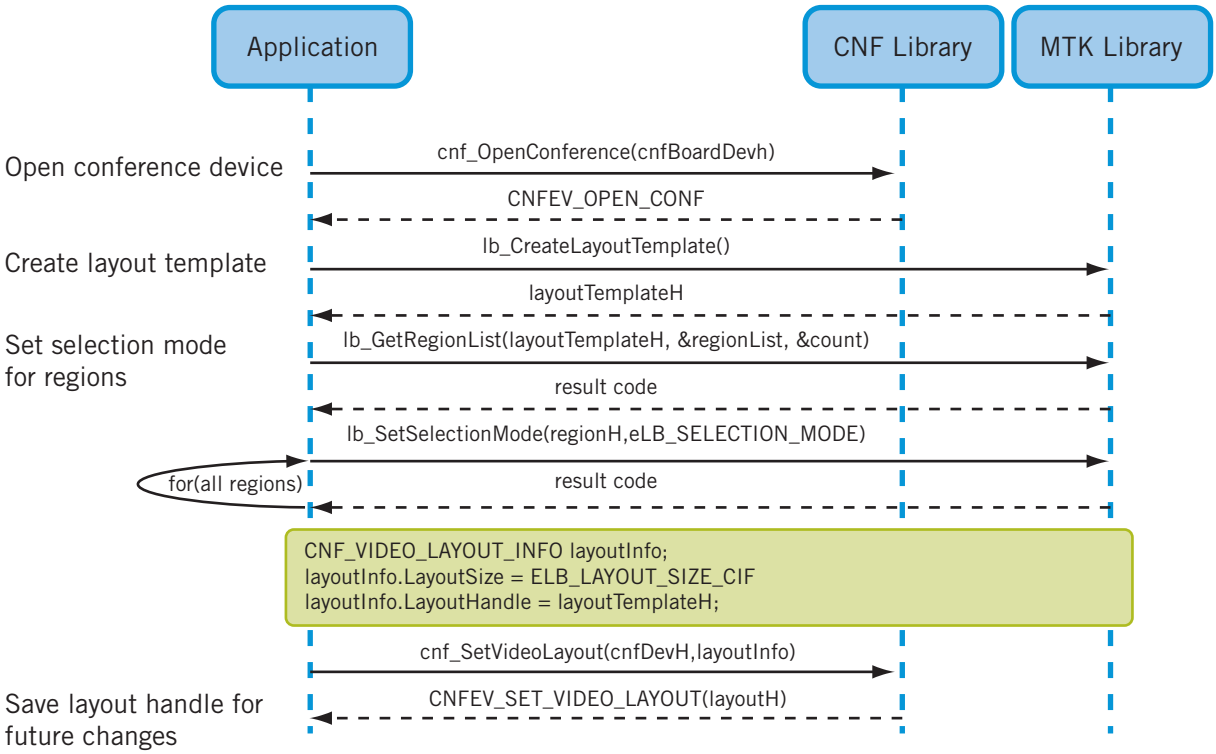

Figure 4. Sequence Diagram for Setting up the Conference Video Layout

Creating a Conference Layout Template

Specifying Layout Type

The lb_CreateLayoutTemplate(type) function returns a handle to a layout template for the layout type specified. The MMP/MMK software supports 1, 4 region layout types as well as custom layouts.

Note: Additional predefined layouts, such as 6 and 9 region layout types, are planned for later releases of the MMP/MMK software.

The layout template is then used to set region properties and then applied to the conference.

Setting Region Properties

After the layout template has been created, properties can be set for each region in the layout by iterating over a list of the regions obtained using the lb_GetRegionList() function. The VideoConfDemo application can explicitly determine which conference party is displayed in each region. This application is enabled by calling the lb_SetSelectionMode (regionHandle, mode) function for each region with mode set to eLB_SELECTION_MODE_USER_SELECT. The default mode is to have the region automatically set using the active talker algorithm.

Applying the Conference Layout Template

The conference layout template is applied to the conference using the cnf_SetVideoLayout() function. The completion event for this function contains a handle for the active layout. This handle is different than the handle for the layout template. Subsequent changes to the layout template can be applied to the conference by the calling cnf_SetVideoLayout() function again.

Adding Parties to the Conference

When a party is added to a conference using the cnf_AddParty() function, the audio and video streams are available for mixing into the conference output. The audio of all conference parties is summed to create the output. The active talker algorithm can be enabled to limit the audio summation to the three loudest talkers. The video from the conference parties cannot just be summed into an output stream; instead, the individual streams have to be combined using the conference video layout. This requires that the video streams for visible conference parties be assigned to the layout regions (see Figure 5).

Video from conference parties are assigned to conference layout regions by creating a CNF_VISIBLE_PARTY_LIST. Passing the list as a parameter to the cnf_SetVisiblePartyList() function applies the region-party assignments and makes them active. The CNF_VISIBLE_PARTY_LIST contains an array of CNF_VISIBLE_PARTY_INFO structures. The CNF_VISIBLE_PARTY_INFO structure requires a handle to a conference party and a handle to a layout region. The application maintains a list of parties that have been added to the conference, and a list of region handles is obtained by using the Layout Builder lb_GetRegionList() function.

Image Overlay

The VideoConfDemo application includes a feature that displays caller identifiers for each party for five seconds when a caller joins or leaves the conference, when the conference layout is changed, or when the visible parties are rotated within the layout. The caller identifier is an image file applied as an overlay to the video stream coming from the caller into the conference.

Image overlays are created using the Overlay Builder, Stream Manipulation, and other MTK APIs. The VideoConfDemo COverlay class contains the API calls to create an overlay and apply it to a video stream. The SipSession objects create an Overlay object and use it to add the caller identifier image overlay to the caller's video stream going to the conference.

An image overlay is applied by creating an image overlay template and then applying the overlay template to a video stream of a device. The image overlay template is created first by using the mtk_CreateImageTemplate() function. JPEG files are used by the VideoConfDemo application for the image source so the mtk_CreateMediaFileTemplate() function is used to specify the JPEG file. A bounding frame for the image is set using a frame template created using the mtk_CreateFrameTemplate() function, and its properties are set with the mtk_SetFramePositionOverlay() and mtk_SetFrameSize() functions. The frame template is applied to the overlay template using the ob_SetOverlayBoundingFrame() function.

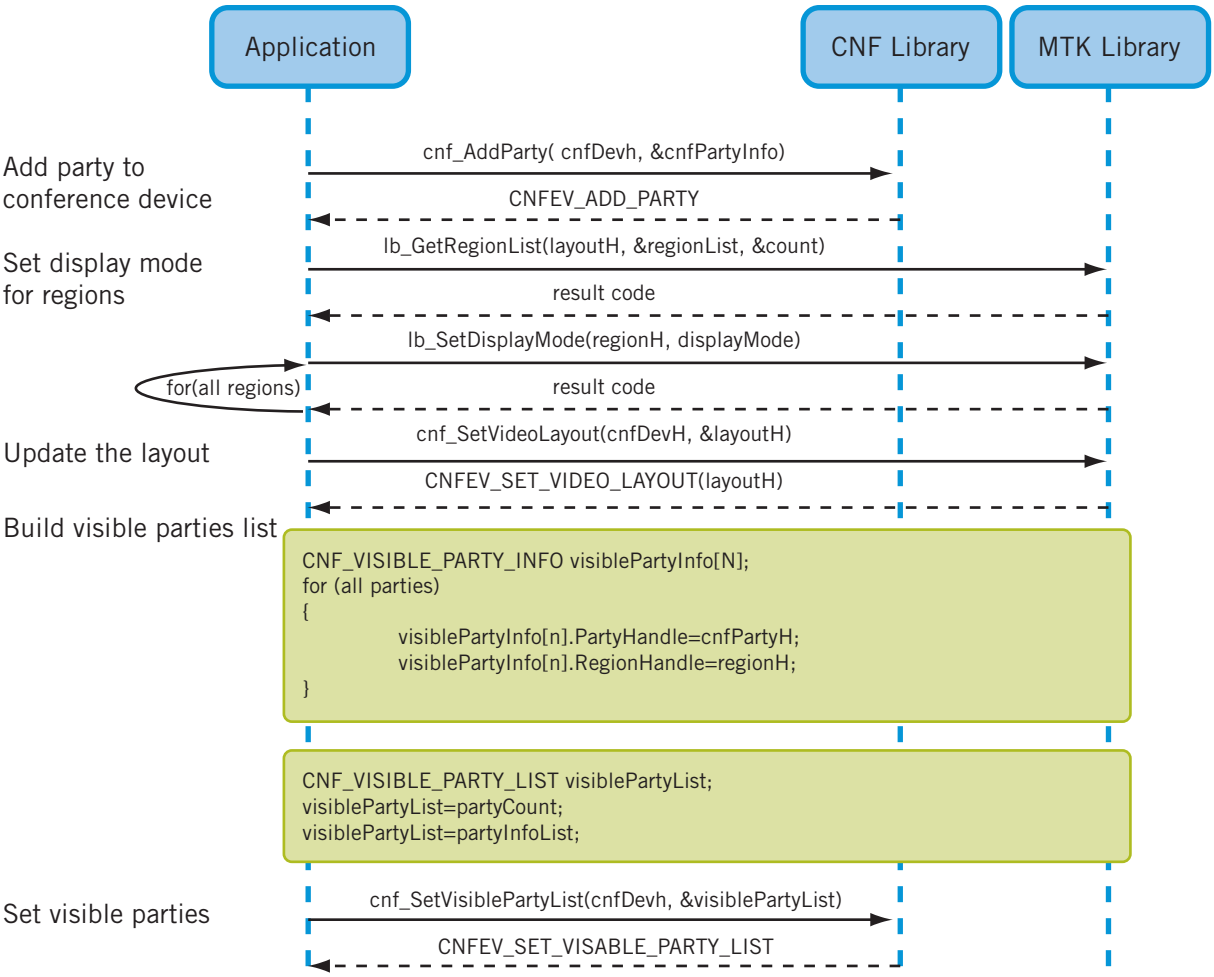

Figure 5. Sequence Diagram for Adding a Party to the Video Conference

After the overlay template is created and all properties have been set, the overlay template is applied to the video stream using the sm_AddOverlays() function (used for stream manipulation). Figure 6 is an example of a caller ID used as an image overlay.

For additional information and a description of how to use image overlays, refer to *Using Dialogic® Media Toolkit API for Image Overlay on Video Streams*, and for information on converting text into images that can be used by the image overlay API, refer to *Using Dialogic® Media Toolkit API and ImageMagick for Image Overlay on Video Streams* (see the *For More Information* section).

Transcoding, Video Transrating, and Image Resizing

Each SIP caller may use a set of supported coders and coder parameters. The coders and parameters are detected as part of call setup coder negotiation, and the IPM device streams are configured appropriately when the RTP stream is started. The Multimedia devices used for playing clips are configured to use coders appropriate to the media files being played. As the coders used by the callers and the media files may be different, transcoding will be required. Also, coder parameters such as video frame rate and resolution format may be different, so video transrating and image resizing may be required.

Figure 6. Caller ID as an Image Overlay

These three functions are performed by the device management system when two devices are connected using the dev_PortConnect() function. No explicit API for enabling transcoding, transrating, and resizing exists, but rather they are automatically performed as a result of using the dev_PortConnect() function to join two multimedia streams. The actual transcoding, transrating, and resizing operations are determined by the media file, MM, IPM, and CNF device settings.

Coder Negotiation

Coder negotiation is performed when an inbound call is offered. The coders proposed in the SIP INVITE message SDP are matched against the local capabilities. If a match is found, the coder resource is reserved and the call is answered. If a match cannot be found, the inbound call is rejected.

A set of classes is used to create coder objects for both the offer SDP and local capabilities. Each supported coder class has the logic to obtain coder-specific parameters from the SIP INVITE message SDP as well as from IPM_AUDIO_CODER_INFO and IPM_VIDEO_CODER_INFO local capabilities structures initialized for the supported coders. The classes also have the logic to compare two coders and to export the SDP for a matched coder.

When an inbound call is offered, the VideoConfDemo SipSession object builds a remote capabilities list and a local capabilities list, and then finds the first match between the two lists. If a match is found and a coder resource can be reserved, then the call is accepted. Otherwise, the call is rejected.

Summary

The video conferencing demo shows how the MMP/MMK software can support advanced video features for multimedia applications, such as video transcoding, video transrating and image resizing, and multimedia conferencing.

This application note also described the features of the VideoConfDemo application, and provided information on how to build, configure, and run the VideoConfDemo application. The high-level design of the application was presented, as was a description of how conferencing and multimedia resources can be used to implement the video features of the application.

References

A Zip file containing the demo application and sample code application can be downloaded at
http://www.dialogic.com/goto/?11167

Using Dialogic® Media Toolkit API for Image Overlay on Video Streams —
http://www.dialogic.com/goto/?11011

Using Dialogic® Media Toolkit API and ImageMagick for Image Overlay on Video Streams —
http://www.dialogic.com/goto/?11033-01

Dialogic® Media Toolkit API Library Reference —
http://www.dialogic.com/manuals/docs/media_toolkit_api_v2.pdf

Dialogic® Multimedia API Library Reference —
http://www.dialogic.com/manuals/docs/multimedia_api_v1.pdf

Dialogic® Multimedia API Programming Guide —
http://www.dialogic.com/manuals/docs/multimedia_programming_lin_v2.pdf

Dialogic® IP Media Library API Library Reference —
http://www.dialogic.com/manuals/docs/ip_media_api_hmp_v8.pdf

Dialogic® IP Media Library API Programming Guide —
http://www.dialogic.com/manuals/docs/ip_media_programming_hmp_v3.pdf

Dialogic® Conferencing API Library Reference —
http://www.dialogic.com/manuals/docs/conferencing_api_v2.pdf

Dialogic® Conferencing API Programming Guide —
http://www.dialogic.com/manuals/docs/conferencing_programming_v2.pdf

Dialogic® Device Management API Library Reference —
http://www.dialogic.com/manuals/docs/device_mgmt_api_v7.pdf

Dialogic® Global Call API Library Reference —
http://www.dialogic.com/manuals/docs/globalcall_api_hmp_v4.pdf

Dialogic® Global Call API Programming Guide —
http://www.dialogic.com/manuals/docs/globalcall_programming_hmp_v4.pdf

Dialogic® Global Call IP Technology Guide —
http://www.dialogic.com/manuals/docs/globalcall_for_ip_hmp_v10.pdf

Kapanga Softphone information —
http://www.kapanga.net

Dialogic®
Making Innovation Thrive™

www.dialogic.com

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9

09/08 111091-01