Dialogic®

Dialogic® Media Toolkit API

Library Reference

August 2011

05-2603-003

Publication Date: August 2011

Document Number: 05-2603-003

Contents

Figures

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-2603-003	August 2011	Function Summary by Category chapter: Added a Media Parsing Functions section. Function Information chapter: Added the *mp_mtklib*.h header file to the text. mp_CloseFile( ): Added. mp_GetFileInfo( ): Added. mp_GetFileInfoSize( ): Added. mp_OpenFile( ): Added. Data Structures chapter: Added new data structures. MP_FILE_INFO_DESC: Added. MP_MMFILE: Added. Error Codes chapter: Added a section about Media Parsing API errors.
05-2603-002	February 2008	Updated to indicate support for Media Toolkit General Purpose functions, Overlay Builder functions, Stream Manipulation functions, and Error Processing functions. Function Summary by Category chapter: Updated to show that Media Toolkit General Purpose functions, Overlay Builder functions, Stream Manipulation functions, and Error Processing functions are now supported. Function Information chapter: Removed ob_GetOverlayBoundingRectangle( ), ob_SetOverlayBoundingRectangle( ), ob_GetOverlayJustification( ), ob_SetOverlayJustification( ), mtk_GetResultInfo( ). lb_CreateLayoutTemplate( ) function: Added eLB_LAYOUT_TYPE_6_1 and eLB_LAYOUT_TYPE_9_1. Added more information on predefined layout types. mtk_CreateFrameTemplate( ) function: Added. mtk_DestroyFrameTemplate( ) function: Added. mtk_GetFramePosition( ) function: Added. mtk_GetFrameSize( ) function: Added. mtk_SetFramePosition( ) function: Added. mtk_SetFrameSize( ) function: Added. ob_GetOverlayBoundingFrame( ) function: Added. ob_GetOverlayFillStyle( ) function: Added ob_SetOverlayBoundingFrame( ) function: Added ob_SetOverlayFillStyle( ) function: Added. sm_AddOverlays( ) function: Added. sm_RemoveAllOverlays( ) function: Added. sm_RemoveOverlays( ) function: Added. Events chapter: Updated with event information. Data Structures chapter: Removed MTK_EVENT_INFO.

Document No.	Publication Date	Description of Revisions
		SM_ADD_OVERLAY data structure: Added. SM_ADD_OVERLAY_LIST data structure: Added. SM_ADD_OVERLAY_RESULT data structure: Added. SM_ADD_OVERLAY_RESULT_LIST data structure: Added. SM_REMOVE_OVERLAY_LIST data structure: Added. Supplementary Reference Information chapter: Added Stream Manipulation Media Toolkit Example Code section.
05-2603-001	October 2007	Initial version of document.

About This Publication

The following topics provide more information about this publication:

Purpose

This publication provides a reference to functions, parameters, and data structures in the Dialogic® Media Toolkit API.

Applicability

This document version is published for the Dialogic® Host Media Processing Software Release 4.1LIN.

This document may also be applicable to other software releases (including service updates) on Linux or Windows® operating systems. Check the Release Guide for your software release to determine whether this document is supported.

Intended Audience

This publication is intended for the following audience:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

How to Use This Publication

This document assumes that you are familiar with the Linux or Windows® operating systems and the C++ programming language.

The information in this document is organized as follows:

- Chapter 1, "Function Summary by Category" introduces the various categories of functions and provides a brief description of each function.
- Chapter 2, "Function Information" provides an alphabetical reference to the functions.
- Chapter 3, "Events" provides an alphabetical reference to events that may be returned by the Dialogic® Media Toolkit API software.
- Chapter 4, "Data Structures" provides an alphabetical reference to the data structures.
- Chapter 5, "Error Codes" presents a list of error codes that may be returned by the Dialogic® Media Toolkit API software.
- Chapter 6, "Supplementary Reference Information" provides reference information including example code of Dialogic® Media Toolkit API functions.

Related Information

Refer to the following sources for more information:

- For information on the software release, system requirements, release features, and release documentation, see the Release Guide for the software release you are using.
- For details on compatibility issues, restrictions and limitations, known issues, and late-breaking updates or corrections to the release documentation, see the Release Update for the software release you are using.
- For Dialogic® product documentation including the Release Guide and the Release Update, see *http://www.dialogic.com/manuals/*.
- For Dialogic technical support, see *http://www.dialogic.com/support/*
- For Dialogic® product information, see *http://www.dialogic.com/*

Function Summary by Category 1

This chapter describes the categories in which the Dialogic® Media Toolkit API library functions can be logically grouped.

1.1 Media Toolkit General Purpose Functions

The Dialogic® Media Toolkit API library consists of a main library (mtk) and several sub-libraries, namely layout builder (lb), overlay builder (ob), stream manipulation (sm), and media parsing (mp). Each library has its own set of functionality.

Media toolkit general purpose functions are used to create and define media-related templates. These templates represent generic media-related items in a multimedia environment, such as images, audio/video/image files, bitmaps, and frames. A frame refers to an area on a video screen in which a media item is displayed.

The attributes of a media template describe the content of a specific media item being referenced by the template. For example, an attribute of a YUV image template is its data format. Another example of a template attribute is the size of a bounding frame.

mtk_CreateBitmapTemplate( )
creates a bitmap template

mtk_CreateFrameTemplate( )
creates a frame template

mtk_CreateImageTemplate( )
creates a media image template

mtk_CreateMediaFileTemplate( )
creates a media file template

mtk_DestroyFrameTemplate( )
destroys a frame template

mtk_DestroyMediaTemplate( )
destroys a media template

mtk_GetBitmapData( )
gets the data for a bitmap template

mtk_GetFramePosition( )
gets the position of a frame template

mtk_GetFrameSize( )
gets the size of a frame template

mtk_GetMediaFileName( )
gets the file name for a media file template

mtk_GetYUVImageFormat( )
gets the color format for a YUV image template

mtk_GetYUVImageSize( )
gets the size for a YUV image template

mtk_SetBitmapData( )
sets the data for a bitmap template

mtk_SetFramePosition( )
sets the position of a frame template

mtk_SetFrameSize( )
sets the size of a frame template

mtk_SetMediaFileName( )
sets the file name for a media file template

mtk_SetYUVImageFormat( )
sets the color format for a YUV image template

mtk_SetYUVImageSize( )
sets the size for a YUV image template

1.2 Layout Builder Functions

Layout builder functions are used to define video layout templates for a multimedia conference. A video layout template can then be applied to a video capable device, such as a multimedia conferencing device. Layout builder functions are used in conjunction with the Dialogic® Conferencing (CNF) API library.

The video layout template describes the layout (a rectangular area) and one or more regions within the layout. Media streaming occurs within a region; therefore, at least one region must be defined for the layout.

The Dialogic® Media Toolkit API library provides predefined layout types. Custom layout types are also supported.

lb_AddRegion( )
adds a region to a video layout

lb_CreateLayoutTemplate( )
creates a video layout template

lb_DestroyLayoutTemplate( )
destroys a video layout template

lb_GetDisplayMode( )
gets the display mode for the region

lb_GetPriority( )
gets the priority of the region

lb_GetRect( )
gets the rectangle for the region

lb_GetRegionList( )
gets the list of regions for the video layout

lb_GetSelectionMode( )
gets the selection mode for the region (such as active talker or user defined)

lb_GetType( )
gets the video layout type (predefined or custom)

lb_RemoveRegion( )
removes a region from the video layout

lb_SetDisplayMode( )
sets the display mode for the region

lb_SetPriority( )
sets the region priority

lb_SetRect( )
sets the rectangle for the region

lb_SetSelectionMode( )
sets the selection mode for the region

1.3 Overlay Builder Functions

Overlay builder functions are used to create and define overlay templates. An overlay template's attributes describe how an overlay is integrated in the media stream for a specific device. After you have defined an overlay template by setting its attributes, the template is used to apply an overlay to a device through the stream manipulation functions.

ob_CreateImageOverlayTemplate( )
creates a media image overlay template

ob_DestroyOverlayTemplate( )
destroys an overlay template

ob_GetOverlayBoundingFrame( )
gets the bounding frame for an overlay template

ob_GetOverlayDuration( )
gets the length of time that an overlay plays over a media stream

ob_GetOverlayFillStyle( )
gets the overlay fill style for an overlay template

ob_SetOverlayBoundingFrame( )
sets the bounding frame for an overlay template

ob_SetOverlayDuration( )
sets the length of time that an overlay plays over a media stream

ob_SetOverlayFillStyle( )
sets the overlay fill style for an overlay template

1.4 Stream Manipulation Functions

Stream manipulation functions are used to manage overlays on a streaming device. The overlays are defined using overlay builder functions.

sm_AddOverlays( )
adds one or more overlays to a device

sm_RemoveAllOverlays( )
removes all overlays from a device

sm_RemoveOverlays( )
removes one or more overlays from a device

1.5 Media Parsing Functions

Media Parsing functions are used to retrieve file information from multimedia files. This file information can be used by the application to:

- decide whether video transcoding/transrating/transsizing is necessary.
- decide whether audio transcoding/transrating is necessary.
- retrieve information necessary for call/media session setup purposes.
- decide which media track to play

Note: Track selection is currently not supported with the Multimedia API.

- Simple dump of information.

The Media Parsing API set consists of the following functions:

mp_CloseFile( )
closes a previously-opened multimedia file

mp_GetFileInfo( )
retrieves detailed file information from a multimedia file

mp_GetFileInfoSize( )
optional function that returns the minimum buffer size needed to pass to the **mp_GetFileInfo( )** function

mp_OpenFile( )
opens a multimedia file for reading

Note: Media Parsing functions do not use the error processing functions that other API libraries use.

The order for calling the Media Parsing APIs to retrieve file information is as follows:

1. **mp_OpenFile( )**
2. **mp_GetFileInfoSize( )**
3. **mp_GetFileInfo( )**
4. **mp_CloseFile( )**

A demo program, mptestdemo, is available to demonstrate Media Parsing API functionality and all of the file information that is retireved. Its source code, along with a readme file containing usage and sample output, is located in the /usr/dialogic/demos/mptestdemo directory. In addition, a precompiled executable, *mptest*, is located in the /usr/dialogic/bin directory.

1.6 Error Processing Functions

Error processing functions get general information and error information associated with an event.

mtk_GetErrorInfo( )
gets error information for a failed function

Function Information 2

This chapter contains a detailed description of each Dialogic® Media Toolkit API function, presented in alphabetical order. A general description of the function syntax is given before the detailed function information.

Function prototypes are defined in the following header files: *mtklib.h*, *lb_mtklib.h*, *ob_mtklib.h*, *sm_mtklib.h* and *mp_mtklib.h*.

2.1 Function Syntax Conventions

The Dialogic® Media Toolkit API functions typically use the following format:

```
datatype xx_Function (deviceHandle, parameter1, parameter2, ... parametern)
```

where:

datatype
: refers to the data type; for example, MTK_RETURN and MTK_DEVICE_HANDLE (see the header files for a definition of data types)

xx_Function
: is the name of the function; xx represents mtk (media toolkit), lb (layout builder), ob (overlay builder), sm (stream manipulation), or mp (media parsing)

deviceHandle
: refers to an input field representing the type of device handle

parameter1, parameter2, ... parameter*n*
: represent input or output fields

lb_AddRegion( )

Name: LB_FRAME_HANDLE lb_AddRegion (a_LayoutHandle, a_pRect)

Inputs: LB_FRAME_HANDLE a_LayoutHandle • layout handle

PMTK a_pRect • pointer to rectangle structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_AddRegion( )** function adds a region to the specified video layout. The function returns a region handle to uniquely identify the added region within the specified layout.

This function is only supported for adding regions to a custom layout type. This function is not supported for predefined layout types. Layout types are identified in the eLB_LAYOUT_TYPE enumeration; see **lb_CreateLayoutTemplate( )**.

Call **lb_RemoveRegion( )** when the application is done with the layout region. Calls to **lb_DestroyLayoutTemplate( )** removes all regions previously added to the layout.

Parameter	Description
a_LayoutHandle	layout handle obtained from **lb_CreateLayoutTemplate( )** or **cnf_GetVideoLayout( )**
a_pRect	pointer to a rectangle data structure, MTK_RECT. For a layout, the position and dimensions of a rectangle are expressed in percentages.

For information on Conferencing (CNF) API functions, see the *Dialogic® Conferencing API Library Reference*.

The function returns an LB_FRAME_HANDLE if the specified handle was successfully returned; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_CreateLayoutTemplate( )**
- **lb_DestroyLayoutTemplate( )**
- **lb_RemoveRegion( )**

lb_CreateLayoutTemplate( )

Name: LB_FRAME_HANDLE lb_CreateLayoutTemplate (a_eType)

Inputs: eLB_LAYOUT_TYPE a_eType
- video layout type

Returns: layout handle if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_CreateLayoutTemplate( )** function creates a new video layout template that can be applied to a device that supports video layouts, such as multimedia conferencing. This function returns a layout handle to uniquely identify the video layout template. All subsequent references to this layout template must be made using the handle returned.

You can customize the layout template and apply it to one or more devices. Applying a layout template to a device will realize that layout template on the given device. To apply a layout template to a multimedia conferencing device, use **cnf_SetVideoLayout( )**. Once a layout template has been applied to a device, subsequent modifications to the layout template using the layout handle will not affect the layout on that device. Subsequent modifications take effect only by reapplying the layout handle to the device.

For information on Conferencing (CNF) API functions, see the *Dialogic® Conferencing API Library Reference*.

Parameter	Description
a_eType	type of video layout. The eLB_LAYOUT_TYPE data type is an enumeration that defines the following values: • eLB_LAYOUT_TYPE_CUSTOM – Custom layout type defined by the user. • eLB_LAYOUT_TYPE_1_1 – Predefined one region layout variation 1; region takes up the entire layout. • eLB_LAYOUT_TYPE_4_1 – Predefined four region layout variation 1; layout is divided in four equal regions. • eLB_LAYOUT_TYPE_6_1 – Predefined six region layout variation 1; consists of one larger region and five smaller regions of equal size on the bottom and right side of the layout. • eLB_LAYOUT_TYPE_9_1 – Predefined nine region layout variation 1; layout is divided in nine equal regions.

This function returns an LB_FRAME_HANDLE if successful; otherwise, it returns MTK_ERROR.

Layout Types

More information on the predefined layout types is provided.

Note: In the layout figures, the numbers are included for descriptive purposes only. They have no effect on the code.

To modify the display mode setting, use **lb_SetDisplayMode( )**. To modify the selection mode setting, use **lb_SetSelectionMode( )**.

One Region Layout (1_1)

For the one region layout (1_1):

- Display mode default is live streaming.
- Selection mode default is voice-activated (active talker).

Four Region Layout (4_1)

For the four region layout (4_1):

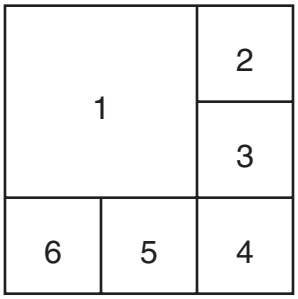
- Display mode default is live streaming for all regions.
- Selection mode default is user select for all regions.

Six Region Layout (6_1)

For the six region layout (6_1):

- Display mode default is live streaming for all regions.
- Selection mode default is voice-activated (active talker) for region 1; user select for regions 2-6.

Figure 1. Six Region Layout (6_1)

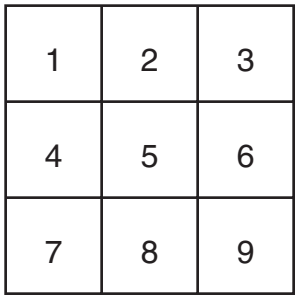


Nine Region Layout (9_1)

For the nine region layout (9_1):

- Display mode default is live streaming for all regions.
- Selection mode default is user select for all regions.

Figure 2. Nine Region Layout (9_1)



Cautions

Be sure to call **lb_DestroyLayoutTemplate( )** when the application is done with the video layout template.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_DestroyLayoutTemplate( )**
- **lb_SetDisplayMode( )**
- **lb_SetSelectionMode( )**

lb_DestroyLayoutTemplate( )

Name: MTK_RETURN lb_DestroyLayoutTemplate (a_LayoutHandle)

Inputs: LB_FRAME_HANDLE a_LayoutHandle • layout handle

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_DestroyLayoutTemplate( )** function destroys a previously created video layout template.

This function releases resources and removes added regions directly associated with the specified layout handle that was created with **lb_CreateLayoutTemplate( )**. After **lb_DestroyLayoutTemplate( )** is called, the associated layout handle is no longer valid for use with any other library functionality.

Destroying a layout template does not impact the devices in which the layout template was applied; that is, the layout template is still present for the affected devices (a snapshot of the template is associated with the device).

Parameter	Description
a_LayoutHandle	handle to a layout template to be destroyed

Cautions

- After this function is called, the associated layout handle and all region handles of the layout are no longer valid for use with any other library functionality.
- Only layout templates created using **lb_CreateLayoutTemplate( )** can be destroyed with this function.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, “Layout Builder Media Toolkit Example Code”, on page 115.

See Also

- **lb_CreateLayoutTemplate( )**

lb_GetDisplayMode( )

Name: MTK_RETURN lb_GetDisplayMode (a_RegionHandle, a_pMode)

Inputs: LB_FRAME_HANDLE a_RegionHandle — region handle

eLB_DISPLAY_MODE * a_pMode — pointer to display mode

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_GetDisplayMode( )** function gets the display mode for the specified region within the video layout.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )** or **lb_GetRegionList( )**
a_pMode	pointer to the display mode

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_SetDisplayMode( )**

lb_GetPriority( )

Name: MTK_RETURN lb_GetPriority (a_RegionHandle, a_punPriority)

Inputs: LB_FRAME_HANDLE a_RegionHandle
- region handle

Outputs: unsigned int * a_punPriority
- pointer to region priority

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_GetPriority( )** function gets the priority for the specified region. The priority value determines which region is displayed in the layout when two or more regions overlap.

If all regions have the same priority, the last one added takes precedence.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )**
a_punPriority	pointer to the region priority set using **lb_SetPriority( )**

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, “Layout Builder Media Toolkit Example Code”, on page 115.

See Also

- **lb_SetPriority( )**

lb_GetRect( )

Name: MTK_RETURN lb_GetRect (a_RegionHandle, a_pRect)

Inputs: LB_FRAME_HANDLE a_RegionHandle
- region handle

Outputs: PMTK a_pRect
- pointer to rectangle structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_GetRect( )** function gets the rectangle for the specified region.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )**
a_pRect	pointer to a rectangle data structure, MTK_RECT. For a layout, the position and dimensions of a rectangle are expressed in percentages.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_SetRect( )**

lb_GetRegionList( )

Name:	MTK_RETURN lb_GetRegionList (a_LayoutHandle, a_pRegionList, a_pRegionCount)	
Inputs:	LB_FRAME_HANDLE a_LayoutHandle	• layout handle
Outputs:	LB_FRAME_HANDLE * a_pRegionList	• pointer to region handle list
Input/Output:	unsigned int * a_pRegionCount	• pointer to region handle count
Returns:	MTK_SUCCESS if successful MTK_ERROR on failure	
Includes:	lb_mtklib.h	
Category:	Layout Builder	
Mode:	synchronous	

Description

The **lb_GetRegionList( )** function gets the list of regions for the specified video layout. The **a_pRegionCount** variable specifies the size of the provided region list. If the actual region count is greater than the provided region count, this function will fail and the **a_pRegionCount** variable will be updated to reflect the actual region count.

Upon successful completion of this function call, **a_pRegionList** and **a_pRegionCount** will contain the video layout region information.

Parameter	Description
a_LayoutHandle	layout handle obtained from **lb_CreateLayoutTemplate( )** or **cnf_GetVideoLayout( )**
a_pRegionList	pointer to region handle list
a_pRegionCount	pointer to region handle count

For information on Conferencing (CNF) API functions, see the *Dialogic® Conferencing API Library Reference*.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_AddRegion( )**
- **lb_CreateLayoutTemplate( )**
- **lb_RemoveRegion( )**

lb_GetSelectionMode( )

Name: MTK_RETURN lb_GetSelectionMode (a_RegionHandle, a_peMode)

Inputs: LB_FRAME_HANDLE a_RegionHandle • region handle

Outputs: eLB_SELECTION_MODE * a_peMode • pointer to selection mode

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_GetSelectionMode( )** function gets the selection mode for the specified region. The selection mode is specified using **lb_SetSelectionMode( )**.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )** or **lb_GetRegionList( )**
a_peMode	pointer to the selection mode

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, “Layout Builder Media Toolkit Example Code”, on page 115.

See Also

- **lb_SetSelectionMode( )**

lb_GetType( )

Name: MTK_RETURN lb_GetType (a_LayoutHandle, a_pType)

Inputs: LB_FRAME_HANDLE a_LayoutHandle
- layout handle

Outputs: eLB_LAYOUT_TYPE * a_peType
- pointer to layout type

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_GetType( )** function gets the layout type for the specified layout, which includes custom layout type and predefined layout types.

Parameter	Description
a_LayoutHandle	layout handle obtained from **lb_CreateLayoutTemplate( )**
a_peType	pointer to the layout type

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_CreateLayoutTemplate( )**

lb_RemoveRegion( )

Name: MTK_RETURN lb_RemoveRegion (a_LayoutHandle, a_RegionHandle)

Inputs: LB_FRAME_HANDLE a_LayoutHandle • layout handle

LB_FRAME_HANDLE a_RegionHandle • region handle

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_RemoveRegion( )** function removes an existing region from a video layout.

This function is only supported for removing a region from a custom layout type. This function is not supported for predefined layout types. Layout types are identified in the eLB_LAYOUT_TYPE enumeration; see **lb_CreateLayoutTemplate( )**.

Parameter	Description
a_LayoutHandle	layout handle obtained from **lb_CreateLayoutTemplate( )** or **cnf_GetVideoLayout( )**
a_RegionHandle	region handle obtained from **lb_AddRegion( )** or **lb_GetRegionList( )**

For information on Conferencing (CNF) API functions, see the *Dialogic® Conferencing API Library Reference*.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **lb_AddRegion( )**
- **lb_GetRegionList( )**

lb_SetDisplayMode( )

Name: MTK_RETURN lb_SetDisplayMode (a_RegionHandle, a_eMode)

Inputs: LB_FRAME_HANDLE a_RegionHandle
- region handle

Outputs: ELB_DISPLAY_MODE a_eMode
- display mode

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_SetDisplayMode( )** function sets the display mode for the specified region.

Each predefined video layout comes with a default display mode. For example, the one-region layout and the four-region layout have live streaming as the default value. See **lb_CreateLayoutTemplate( )** for more on default values.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )** or **lb_GetRegionList( )**
a_eMode	display mode for the region. The ELB_DISPLAY_MODE data type is an enumeration that defines the following values: • eLB_DISPLAY_MODE_BLANK – No video is displayed; instead, background color is displayed in the region. • eLB_DISPLAY_MODE_FROZEN – Display static video (snapshot of live capture image) of the participant in the region. • eLB_DISPLAY_MODE_LIVE – Display live streaming of the participant in the region.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, “Layout Builder Media Toolkit Example Code”, on page 115.

See Also

- **lb_CreateLayoutTemplate( )**
- **lb_GetDisplayMode( )**

lb_SetPriority( )

Name: MTK_RETURN lb_SetPriority (a_RegionHandle, a_unPriority)

Inputs: LB_FRAME_HANDLE a_RegionHandle
- region handle

Outputs: unsigned int a_unPriority
- region priority

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_SetPriority( )** function sets the priority for the specified region. The priority value determines which region is displayed in the layout when two or more regions overlap.

If all regions have the same priority, the last one added takes precedence.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )**
a_unPriority	priority setting for the region. Valid values are 0 to 10, where 0 is disabled and 1 is the highest priority. Default value is 1.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, “Layout Builder Media Toolkit Example Code”, on page 115.

See Also

- **lb_GetPriority( )**

lb_SetRect( )

Name: MTK_RETURN lb_SetRect (a_RegionHandle, a_pRect)

Inputs: LB_FRAME_HANDLE a_RegionHandle
- region handle

Outputs: PMTK a_pRect
- pointer to rectangle structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_SetRect( )** function sets the rectangle for the specified region.

This function is only supported for a custom layout type. This function is not supported for predefined layout types. Layout types are identified in the eLB_LAYOUT_TYPE enumeration; see **lb_CreateLayoutTemplate( )**.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )** or **lb_GetRegionList( )**
a_pRect	pointer to a rectangle data structure, MTK_RECT. For a layout, the position and dimensions of a rectangle are expressed in percentages.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.1, “Layout Builder Media Toolkit Example Code”, on page 115.

See Also

- **lb_GetRect( )**

lb_SetSelectionMode( )

Name: MTK_RETURN lb_SetSelectionMode (a_RegionHandle, a_eMode)

Inputs: LB_FRAME_HANDLE a_RegionHandle • region handle

Outputs: eLB_SELECTION_MODE a_eMode • selection mode

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: lb_mtklib.h

Category: Layout Builder

Mode: synchronous

Description

The **lb_SetSelectionMode( )** function sets the visible party selection mode for the specified region. This mode determines which party is made visible in each region, and how parties in each region are updated, refreshed or removed.

Each predefined video layout comes with a default selection mode. For example, the one-region layout has voice activated mode as the default value; the four-region layout has user select mode as the default value. See **lb_CreateLayoutTemplate( )** for more on default values.

Parameter	Description
a_RegionHandle	region handle obtained from **lb_AddRegion( )** or **lb_GetRegionList( )**
a_eMode	selection mode for the region. The eLB_SELECTION_MODE data type is an enumeration that defines the following values: • eLB_SELECTION_MODE_VOICE_ACTIVATED – In the region, display the current active talker or one of the current active talkers. This mode allows all participants to view the current speaker(s). • eLB_SELECTION_MODE_USER_SELECT – In the region, the application developer specifies the participant to be displayed. The participants to be displayed are specified using the **cnf_SetVisiblePartyList( )** function. For information on Conferencing (CNF) API functions, see the *Dialogic® Conferencing API Library Reference*.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.1, "Layout Builder Media Toolkit Example Code", on page 115.

See Also

- **lb_CreateLayoutTemplate( )**
- **lb_GetSelectionMode( )**

mp_CloseFile( )

Name: MTK_RETURN mp_CloseFile(pMMFile)

Inputs: MP_MMFILE* pMMFILE • pointer to MP_MMFILE structure

Returns: MTK_SUCCESS if successful
MPERR_* (see below) on failure

Includes: mp_mtklib.h

Category: Media Parsing

Mode: synchronous

Description

The **mp_CloseFile( )** function closes a previously-opened multimedia file.

Parameter	Description
pMMFile	pointer to MP_MMFILE structure (input).

Cautions

- Be sure to call this function for every opened multimedia file once the file information has been retrieved.
- Do not call **mtk_GetErrorInfo( )** for error information.

Errors

The function returns MTK_SUCCESS if the multimedia file was successfully closed; otherwise, it returns one of the following errors:

MPERR_ INVALID_FILE
pMMFile invalid

MPERR_ INTERNAL_RESOURCE
internal error

Example

For example code, see the mptestdemo in /usr/dialogic/demos.

See Also

- **mp_OpenFile( )**

mp_GetFileInfo( )

Name: MTK_RETURN mp_GetFileInfo(pMMFile, pFileInfoDesc)

Inputs: MP_MMFILE* pMMFILE
- pointer to MP_MMFILE structure

Outputs: MP_FILE_INFO_DESC* pFileInfoDesc
- pointer to MP_FILE_INFO_DESC structure

Returns: MTK_SUCCESS if successful
MPERR_* (see below) on failure

Includes: mp_mtklib.h

Category: Media Parsing

Mode: synchronous

Description

The **mp_GetFileInfo( )** function returns detailed file information from a previously-opened multimedia file. See **mp_OpenFile( )** for a list of supported file types.

Parameter	Description
pMMFile	pointer toMP_MMFILE structure (input)
pFileInfoDesc	pointer to MP_FILE_INFO_DESC structure (output)

Retrieving the File Information

The pFileInfoDesc structure argument is used for retrieving the file information. The application is responsible for creating a MP_FILE_INFO_DESC structure and passing it to the mp_GetFileInfo( ) function. There are only three structure members that are inputs to the function, the rest of the structure members contain file-level information retrieved from the multimedia file. Refer to the mptestdemo in /usr/dialogic/demos for example code.

As an input, pFileInfo is a pointer to an unformatted allocated block of memory. As an output, it returns the detailed file information, formatted with a set of structures, MP_BLK_* as defined in mp_mtklib.h. The first block of information will be a presentation block (MP_BLK_PRESENTATION), and this is usually followed by one or more media stream blocks, with possible hint stream blocks in between.

Each media stream block usually contains a media codec block. Each block of information begins with a block size field and a block type field. These can be used to determine the type of each block

and to traverse/navigate each block. The following diagram shows a usual configuration of the formatted file information block.

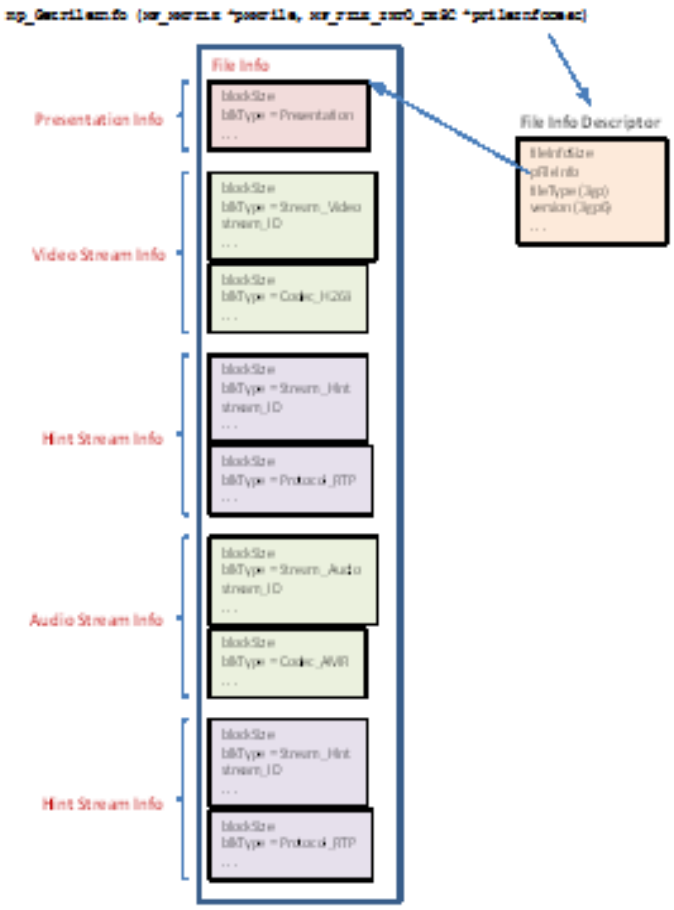


Figure 3. Media Parsing File Information Block – General Structure

The following diagram provides details for each of the blocks. Please refer to mp_mtklib.h for a full description of the parameters in each structure.

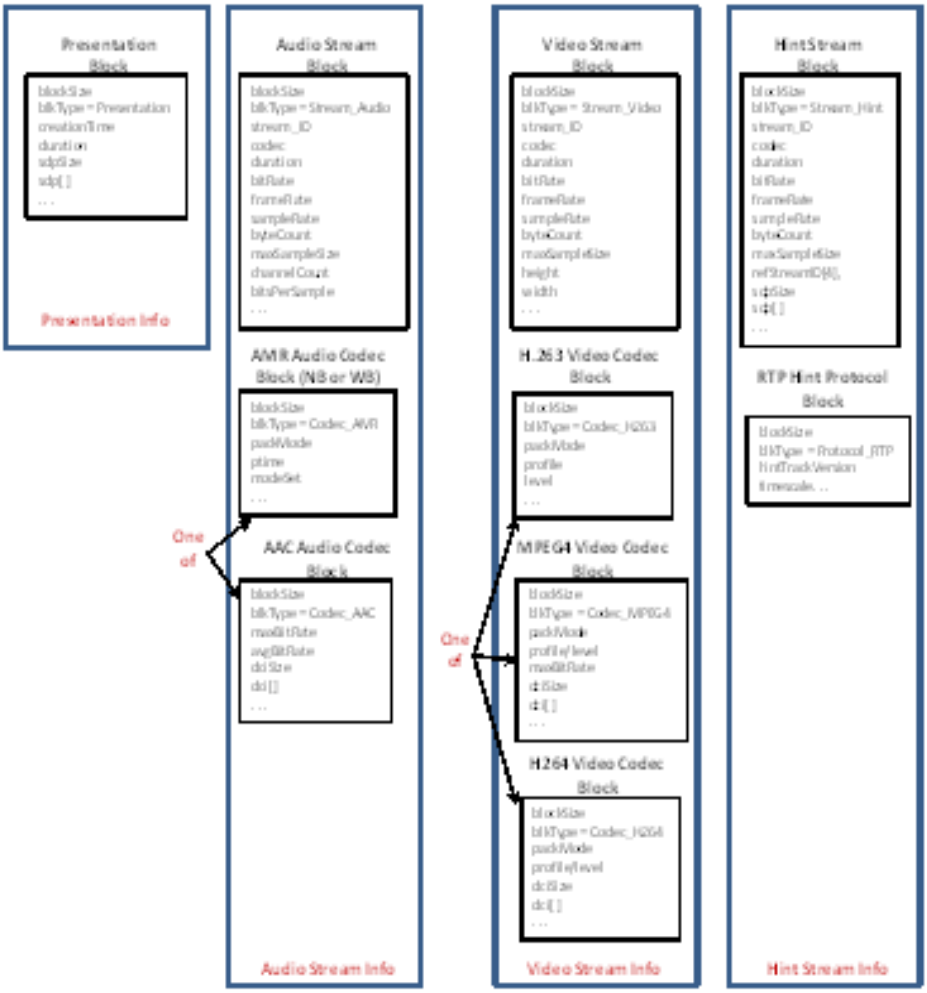


Figure 4. Media Parsing Info Blocks

Note: The demo program demonstrates the Media Parsing functionality. The source code, along with a readme file containing a full description, usage, and sample output is located in /usr/dialogic/demos/mptestdemo. A pre-compiled executable, mptest, is located in /usr/dialogic/bin.

■ Cautions

- Do not call **mtk_GetErrorInfo( )** for error information.

Errors

The function returns MTK_SUCCESS if the retrieval of file information was successful; otherwise, it returns one of the following errors:

MPERR_ INVALID_FILE
pMMFile invalid

MPERR_ INVALID_POINTER
NULL pFileInfoDesc

MPERR_INCOMP_ACCESS_MODE
file not opened for read

MPERR_INVALID_STRUCT_VERSION
invalid version of pFileInfoDesc

Example

For example code, see the mptestdemo in /usr/dialogic/demos.

See Also

- **mp_OpenFile( )**
- **mp_GetFileInfoSize( )**

mp_GetFileInfoSize( )

Name: MTK_RETURN mp_GetFileInfoSize(pMMFile, pMinFileInfoSize)

Inputs: MP_MMFILE* pMMFILE • pointer to MP_MMFILE structure

unsigned long* pMinFileInfoSize • pointer to the returned minimum file information size

Returns: MTK_SUCCESS if successful
MPERR_* (see below) on failure

Includes: mp_mtklib.h

Category: Media Parsing

Mode: synchronous

Description

The m**p_GetFileInfoSize( )** function returns the minimum size needed in a file information block. in order to return the full complement of detailed file information from a previously-opened multimedia file. (See **mp_GetFileInfo( )**.)

Parameter	Description
pMMFile	pointer to MP_MMFILE structure (input). See **mp_OpenFile( )** for a structure description.
pMinFileInfoSize	pointer to the returned minimum file information size (output). This can then be used when setting fileInfoSize inside the MP_FILE_INFO_DESC structure when calling **mp_GetFileInfo( )**.

Cautions

- Do not call mtk_GetErrorInfo( ) for error information.

Errors

The function returns MTK_SUCCESS if successful; otherwise, it returns one of the following errors:

MPERR_ INVALID_FILE
pMMFile invalid

MPERR_ INVALID_POINTER
Null pFileInfoDesc

MPERR_INCOMP_ACCESS_MODE
file not opened for read

Example

For example code, see the mptestdemo in /usr/dialogic/demos.

See Also

- **mp_OpenFile( )**
- **mp_GetFileInfo( )**

mp_OpenFile( )

Name: MTK_RETURN mp_OpenFile(pFileName, fileType, pMMFile)

Inputs: char* pFileName • file name string

unsigned filetype • file type indicator

Input/Output: MP_MMFILE* pMMFILE • pointer to MP_MMFILE structure

Returns: MTK_SUCCESS if successful
MPERR_* (see below) on failure

Includes: mp_mtklib.h

Category: Media Parsing

Mode: synchronous

Description

The **mp_OpenFile( )** opens a multimedia file for reading. This function must be called first before retrieving file information using the mp_GetFileInfo( ). The multimedia files supported are ISO-based files (i.e., 3GP/MP4), as well as Dialogic proprietary (DMF) files.I

Parameter	Description
pFileName	pointer to a text string containing the name of the file to open (input).
fileType	indicates the type of file (input). The fileType values can be one of the following: • MP_FILE_TYPE_ISO – / /Generic ISO-based format file • MP_FILE_TYPE_3GP – 2 //3GP format file (ISO-based) • MP_FILE_TYPE_MP4 – 3 //MP4 format file (ISO-based) • MP_FILE_TYPE_F4V – 4 //Flash H.264 file (ISO-based) • MP_FILE_TYPE_3G2 – 5 //3GPP2 format file (ISO-based) • MP_FILE_TYPE_AVC – 6 //AVC format file (ISO-based) • MP_FILE_TYPE_DMF – 10 //Dialogic Media format file (Dialogic-proprietary .
pMMFile	pointer to MP_MMFILE structure (input/output)

Note: For ISO-based files, it does not matter which file type is used as an input; they are interchangeable. However, the specific file type will be returned in the MP_FILE_INFO_DESC structure after calling **mp_GetFileInfo( )** function. The Dialogic proprietary files must specifically be designated with MP_FILE_TYPE_DMF.

Cautions

- This function must be called first prior to retrieving file information.
- Do not call **mtk_GetErrorInfo( )** for error information.

Errors

The function returns MTK_SUCCESS if the multimedia file was successfully opened; otherwise, it returns one of the following errors:

MPERR_ INVALID_FILE_TYPE
file type not supported

MPERR_ INVALID_POINTER
NULL pMMFile or pFileName

MPERR_FILE_OPEN_FAILED
file not found or open failed

MPERR_ FILE_ACCESS_DENIED
access denied

MPERR_FILE_FORMAT
file format error

Example

For example code, see the mptestdemo in /usr/dialogic/demos.

See Also

- **mp_CloseFile( )**
- **mp_GetFileInfo( )**

mtk_CreateBitmapTemplate( )

Name: MTK_HANDLE mtk_CreateBitmapTemplate (a_hMedia)

Inputs: MTK_HANDLE a_hMedia
- media handle

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_CreateBitmapTemplate( )** function creates a representation or template of a bitmap containing the specified media. The function returns a handle to a bitmap template for use with other API functions when bitmap-related specifications are needed.

Set and get the bitmap data and length of data using **mtk_SetBitmapData( )** and **mtk_GetBitmapData( )**.

Call **mtk_DestroyMediaTemplate( )** when the application is done with the bitmap template.

Parameter	Description
a_hMedia	media handle returned by **mtk_CreateImageTemplate( )**

The function returns an MTK_HANDLE if the bitmap template was successfully created; otherwise, it returns MTK_ERROR.

Cautions

Be sure to call **mtk_DestroyMediaTemplate( )** when the application is done with the bitmap template.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateImageTemplate( )**
- **mtk_DestroyMediaTemplate( )**
- **mtk_GetBitmapData( )**
- **mtk_SetBitmapData( )**

mtk_CreateFrameTemplate( )

Name: MTK_FRAME_HANDLE mtk_CreateFrameTemplate ( )

Inputs: None

Returns: frame handle if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_CreateFrameTemplate( )** function creates a new frame template that is used to describe a rectangular region on the video screen. To specify the size and position of the region, use **mtk_SetFrameSize( )** and **mtk_SetFramePosition( )**.

This function returns an MTK_FRAME_HANDLE if the frame template was successfully created; otherwise, this function returns MTK_ERROR. Call **mtk_DestroyFrameTemplate( )** when the application is done with the frame.

The MTK_FRAME_HANDLE is used in **ob_SetOverlayBoundingFrame( )** after you have set the size and position attributes.

Cautions

Call **mtk_DestroyFrameTemplate( )** when the application is done with the frame.

Errors

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_DestroyFrameTemplate( )**
- **mtk_GetFramePosition( )**
- **mtk_GetFrameSize( )**
- **mtk_SetFramePosition( )**
- **mtk_SetFrameSize( )**

mtk_CreateImageTemplate( )

Name: MTK_HANDLE mtk_CreateImageTemplate (a_eImageFormat)

Inputs: eMTK_IMAGE_FORMAT a_eImageFormat • image format

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_CreateImageTemplate( )** function creates a representation or template of the media image to be used in the application. The function returns a handle to the media image template for use with other API functions when image-related specifications are needed.

Set the attributes and retrieve the attributes of an image type using various media toolkit image functions. Call **mtk_DestroyMediaTemplate( )** when the application is done with the media image template.

Parameter	Description
a_eImageFormat	media image format. The eMTK_IMAGE_FORMAT data type is an enumeration which defines the following values: • eMTK_IMAGE_FORMAT_YUV – YUV format • eMTK_IMAGE_FORMAT_JPEG – JPEG format ***Note:*** Only YUV 4:2:0 format is currently supported.

The function returns MTK_HANDLE if the image template was successfully created; otherwise, it returns MTK_ERROR.

Cautions

Be sure to call **mtk_DestroyMediaTemplate( )** when the application is done with the media image template.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **mtk_GetYUVImageFormat( )**
- **mtk_GetYUVImageSize( )**
- **mtk_DestroyMediaTemplate( )**
- **mtk_SetYUVImageFormat( )**
- **mtk_SetYUVImageSize( )**

mtk_CreateMediaFileTemplate( )

Name: MTK_HANDLE mtk_CreateMediaFileTemplate (a_hMedia, a_szFileName)

Inputs: MTK_HANDLE a_hMedia
- media handle

const char * a_szFileName
- media file name

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_CreateMediaFileTemplate( )** function creates a template for a media file containing the specified media type. The template contains a snapshot of the information in the template associated with the media handle (**a_hMedia**). The function returns a handle to a media file template for use with other API functions when file-related specifications are needed.

The file name of the media file template associated with the returned handle can be changed by calling **mtk_SetMediaFileName( )**. You can retrieve the current file name value by calling **mtk_GetMediaFileName( )**.

Call **mtk_DestroyMediaTemplate( )** when the application is done with the media file template.

Parameter	Description
a_hMedia	media handle returned by **mtk_CreateImageTemplate( )**
a_szFileName	media file name

The function returns an MTK_HANDLE if the specified file template was successfully created; otherwise, it returns MTK_ERROR.

Cautions

Be sure to call **mtk_DestroyMediaTemplate( )** when the application is done with the media file.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **mtk_CreateImageTemplate( )**
- **mtk_DestroyMediaTemplate( )**
- **mtk_GetMediaFileName( )**
- **mtk_SetMediaFileName( )**

mtk_DestroyFrameTemplate( )

Name: MTK_RETURN mtk_DestroyFrameTemplate (a_hFrame)

Inputs: MTK_FRAME_HANDLE a_hFrame
- frame handle

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_DestroyFrameTemplate( )** function destroys a frame template.

This function releases resources associated with the specified handle (**a_hFrame**) that was returned by **mtk_CreateFrameTemplate( )**. After this function is called, the associated handle is no longer valid for use with any other library functionality.

Parameter	Description
a_hFrame	handle to frame template to be destroyed

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

- After this function call, the associated handle is no longer valid for use with any other library functionality.
- Do not use this function on handles returned from **ob_SetOverlayBoundingFrame( )** or other frame functions that generate snapshots.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateFrameTemplate( )**

mtk_DestroyMediaTemplate( )

Name: MTK_RETURN mtk_DestroyMediaTemplate (a_hMedia)

Inputs: MTK_HANDLE a_hMedia
- handle to a media template to be destroyed

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_DestroyMediaTemplate( )** function destroys a media template.

This function releases resources directly associated with the specified media handle that was created with **mtk_CreateBitmapTemplate( )**, **mtk_CreateImageTemplate( )**, and **mtk_CreateMediaFileTemplate( )** functions. After this function is called, the associated media handle is no longer valid for use with any other library functionality.

Parameter	Description
a_hMedia	handle to a media template to be destroyed

The function returns MTK_SUCCESS if the specified media template was successfully destroyed; otherwise, it returns MTK_ERROR.

Cautions

- After this function call, the associated media handle is no longer valid for use with any other library functionality.
- Do not use this function on handles returned from **mtk_CreateFrameTemplate( )**.

Errors

This function will fail if an invalid media handle is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateMediaFileTemplate( )**
- **mtk_CreateBitmapTemplate( )**
- **mtk_CreateImageTemplate( )**

mtk_GetBitmapData( )

Name: MTK_RETURN mtk_GetBitmapData (a_hBitmap, a_pData)

Inputs: MTK_HANDLE a_hBitmap — • handle to bitmap template

PMTK_BITMAP_DATA a_pData — • pointer to bitmap data structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_GetBitmapData( )** function gets the bitmap data and length of the bitmap template.

Parameter	Description
a_hBitmap	media handle returned by **mtk_CreateBitmapTemplate( )** of the bitmap template for which to get the data
a_pData	pointer to bitmap data structure, MTK_BITMAP_DATA, in which to store the current bitmap data settings

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

This function will fail if an invalid handle or null pointer is specified. Call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateBitmapTemplate( )**
- **mtk_SetBitmapData( )**

mtk_GetErrorInfo( )

Name: MTK_RETURN mtk_GetErrorInfo (a_pErrorInfo)

Inputs: PMTK_ERROR_INFO a_pErrorInfo
- pointer to error information structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Error Processing

Mode: synchronous

Description

The **mtk_GetErrorInfo( )** function obtains error information about a failed function and provides it in the MTK_ERROR_INFO structure. To retrieve the error information, call this function immediately after a function has failed.

Parameter	Description
a_pErrorInfo	pointer to error information structure; for more information, see MTK_ERROR_INFO, on page 105

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

None.

mtk_GetFramePosition( )

Name:	MTK_RETURN mtk_GetFramePosition (a_hFrame, a_px, a_py, a_pePositionType)	
Inputs:	MTK_FRAME_HANDLE a_hFrame	• frame handle
Outputs:	float * a_px	• pointer to horizontal position
	float * a_py	• pointer to vertical position
	eMTK_POSITION_TYPE * a_pePositionType	• position type
Returns:	MTK_SUCCESS if successful MTK_ERROR on failure	
Includes:	mtklib.h	
Category:	Media Toolkit	
Mode:	synchronous	

Description

The **mtk_GetFramePosition( )** function gets the position currently set in a frame template.

Parameter	Description
a_hFrame	handle to frame template for which to get the position
a_px	pointer to a float to be filled with the current horizontal position relative to the video screen
a_py	pointer to a float to be filled with the current vertical position relative to the video screen
a_pePositionType	pointer to eMTK_POSITION_TYPE to be filled with the current type of values stored in **a_px** and **a_py** parameters

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

This function will fail if an invalid handle or a NULL pointer is specified in the call.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateFrameTemplate( )**
- **mtk_GetFrameSize( )**
- **mtk_SetFramePosition( )**
- **mtk_SetFrameSize( )**

mtk_GetFrameSize( )

Name:	MTK_RETURN mtk_GetFrameSize (a_hFrame, a_pw, a_ph, a_peSizeType)	
Inputs:	MTK_FRAME_HANDLE a_hFrame	• frame handle
Outputs:	float * a_pw	• pointer to horizontal size
	float * a_ph	• pointer to vertical size
	eMTK_SIZE_TYPE * a_peSizeType	• size type
Returns:	MTK_SUCCESS if successful MTK_ERROR on failure	
Includes:	mtklib.h	
Category:	Media Toolkit	
Mode:	synchronous	

Description

The **mtk_GetFrameSize( )** function gets the size currently set in a frame template.

Parameter	Description
a_hFrame	handle to frame template for which to get the size
a_pw	pointer to a float to be filled with the current horizontal size
a_ph	pointer to a float to be filled with the current vertical size
a_peSizeType	pointer to eMTK_SIZE_TYPE to be filled with the current type of values stored in **a_pw** and **a_ph** parameters

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

This function will fail if an invalid handle or a NULL pointer is specified in the call.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateFrameTemplate( )**
- **mtk_SetFrameSize( )**
- **mtk_GetFramePosition( )**
- **mtk_SetFramePosition( )**

mtk_GetMediaFileName( )

Name: MTK_RETURN mtk_GetMediaFileName (a_hMediaFile, a_pFileName)

Inputs: MTK_HANDLE a_hMediaFile
- handle of media file item

PMTK_STRING a_pFileName
- pointer to string data structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_GetMediaFileName( )** function gets the file name for a media template.

You must allocate a buffer for the system to store the filename data and provide a pointer in the szString field of the MTK_STRING structure. You must also specify the length of that buffer in the unLength field of this structure. Upon success, the returned unLength is set to the length of the current filename.

Parameter	Description
a_hMediaFile	media handle returned by **mtk_CreateMediaFileTemplate( )** of the media file for which to get the filename
a_pFileName	pointer to string data structure, MTK_STRING, into which the current file name is stored

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

The function fails if the length of the allocated buffer is not adequate to store the current file name and the null terminator. Upon failure, the unLength field in the MTK_STRING structure is set to the length of the current file name. Note that the returned length does not include the null terminator.

Errors

This function will fail if an invalid handle or null pointer is specified. Call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **mtk_CreateBitmapTemplate( )**
- **mtk_CreateMediaFileTemplate( )**

mtk_GetYUVImageFormat( )

Name: MTK_RETURN mtk_GetYUVImageFormat (a_hYUVImage, a_peFormat)

Inputs:

MTK_HANDLE a_hYUVImage	• handle to a YUV image template
eMTK_YUV_IMAGE_FORMAT * a_peFormat	• pointer to image format

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_GetYUVImageFormat( )** function gets the color format currently set in a YUV image template.

Parameter	Description
a_hYUVImage	handle to a YUV image template returned from **mtk_CreateImageTemplate( )** for which to retrieve the image format
a_peFormat	pointer to a variable in which the format of the YUV image template is stored

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

The function will fail if an invalid handle or null pointer is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_SetYUVImageFormat( )**
- **mtk_CreateImageTemplate( )**

mtk_GetYUVImageSize( )

Name:	MTK_RETURN mtk_GetYUVImageSize (a_hYUVImage, a_punWidth, a_punHeight)	
Inputs:	MTK_HANDLE a_hYUVImage	• handle to a YUV image template
	unsigned int * a_punWidth	• pointer to width
	unsigned int * a_punHeight	• pointer to height
Returns:	MTK_SUCCESS if successful MTK_ERROR on failure	
Includes:	mtklib.h	
Category:	Media Toolkit	
Mode:	synchronous	

Description

The **mtk_GetYUVImageSize( )** function gets the dimensions of the YUV image template.

Parameter	Description
a_hYUVImage	handle to a YUV image template returned from **mtk_CreateImageTemplate( )** for which to retrieve the current image size
a_punWidth	pointer to a variable in which the current YUV image template width is stored
a_punHeight	pointer to a variable in which the current YUV image template height is stored

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

The function will fail if an invalid handle or null pointer is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateImageTemplate( )**
- **mtk_SetYUVImageSize( )**

mtk_SetBitmapData( )

Name: MTK_RETURN mtk_SetBitmapData (a_hBitmap, a_pData)

Inputs: MTK_HANDLE a_hBitmap
- handle to bitmap template

PMTK_BITMAP_DATA a_pData
- pointer to bitmap data structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_SetBitmapData( )** function sets the bitmap data for the bitmap template.

Since bitmaps can be very large, the data set by this call must be persistent until the bitmap template and all other media toolkit templates and snapshots referencing it, such as a video overlay, are destroyed or removed. The system does not make a copy of the data, but stores the value in the pointer.

Parameter	Description
a_hBitmap	media handle returned by **mtk_CreateBitmapTemplate( )** of the bitmap template for which to set the data
a_pData	pointer to bitmap data structure, MTK_BITMAP_DATA, which stores the current bitmap data settings

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

The void pointer passed to the function must reference persistent data.

Errors

This function will fail if an invalid handle or null pointer is specified. Call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateBitmapTemplate( )**
- **mtk_GetBitmapData( )**

mtk_SetFramePosition( )

Name: MTK_RETURN mtk_SetFramePosition (a_hFrame, a_x, a_y, a_ePositionType)

Inputs:

MTK_FRAME_HANDLE a_hFrame	• frame handle
float a_x	• horizontal position
float a_y	• vertical position
eMTK_POSITION_TYPE a_ePositionType	• position type

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_SetFramePosition( )** function specifies the horizontal and vertical position of a frame template relative to a base frame such as a video screen. This frame template is created by **mtk_CreateFrameTemplate( )**.

Specify the type of position values in the x and y coordinates by setting the position type. All coordinates use the upper, left-hand corner of the video screen as the origin.

Parameter	Description
a_hFrame	handle to frame template for which to set the position
a_x	horizontal position relative to the video screen
a_y	vertical position relative to the video screen

a_ePositionType	the type of value specified in **a_x** and **a_y** parameters. The eMTK_POSITION_TYPE data type is an enumeration which defines the following values:
	• eMTK_POSITION_TYPE_PIXEL – Pixel-based coordinates; (0,0) is upper, left-hand corner.
	• eMTK_POSITION_TYPE_PERCENT – Percentage-based coordinates; (0,0) is upper, left-hand corner. Values can be up to two positions of precision.
	• eMTK_POSITION_TYPE_JUSTIFICATION – Justification-based coordinates. Supported values for eMTK_JUSTIFY are:
	  • MTK_JUSTIFY_NONE – No justification performed.
	  • MTK_JUSTIFY_LEFT – Left-justified.
	  • MTK_JUSTIFY_RIGHT – Right-justified.
	  • MTK_JUSTIFY_CENTER – Center-justified, either vertically or horizontally.
	  • MTK_JUSTIFY_TOP – Top-justified.
	  • MTK_JUSTIFY_BOTTOM – Bottom-justified.

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information. For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_SetFrameSize( )**
- **mtk_CreateFrameTemplate( )**
- **mtk_GetFramePosition( )**

mtk_SetFrameSize( )

Name: MTK_RETURN mtk_SetFrameSize (a_hFrame, a_w, a_h, a_eSizeType)

Inputs:	MTK_FRAME_HANDLE a_hFrame	• frame handle
	float a_w	• horizontal position
	float a_h	• vertical position
	eMTK_SIZE_TYPE a_eSizeType	• size type

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_SetFrameSize( )** function specifies the size of a frame template relative to a base frame such as a video screen. This frame template is created by **mtk_CreateFrameTemplate( )**.

Parameter	Description
a_hFrame	handle to frame template for which to set the size
a_w	horizontal size
a_h	vertical size
a_eSizeType	the type of value specified in **a_w** and **a_h** parameters. The eMTK_SIZE_TYPE data type is an enumeration which defines the following values: • eMTK_SIZE_TYPE_PIXEL – Pixel-based size. • eMTK_SIZE_TYPE_PERCENT – Percentage-based coordinates; (0,0) is upper, left-hand corner.

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **mtk_CreateFrameTemplate( )**
- **mtk_GetFrameSize( )**
- **mtk_SetFramePosition( )**

mtk_SetMediaFileName( )

Name: MTK_RETURN mtk_SetMediaFileName (a_hMediaFile, a_pFileName)

Inputs: MTK_HANDLE a_hMediaFile
- handle of a media template

PMTK_STRING a_pFileName
- pointer to string data structure

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_SetMediaFileName( )** function sets the file name for a media file template and allows this template to be re-used in another context.

Parameter	Description
a_hMediaFile	media handle returned by **mtk_CreateMediaFileTemplate( )** of the media template for which to set the file name
a_pFileName	pointer to string data structure, MTK_STRING, which stores the current file name

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

This function will fail if an invalid handle or null pointer is specified. Call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateMediaFileTemplate( )**
- **mtk_GetMediaFileName( )**

mtk_SetYUVImageFormat( )

Name: MTK_RETURN mtk_SetYUVImageFormat (a_hYUVImage, a_eFormat)

Inputs: MTK_HANDLE a_hYUVImage — • handle to a YUV image template

eMTK_YUV_IMAGE_FORMAT a_eFormat — • image format value

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

■ Description

The **mtk_SetYUVImageFormat( )** function sets the color format for a YUV image template.

Parameter	Description
a_hYUVImage	handle to a YUV image template returned from **mtk_CreateImageTemplate( )** for which to set the image format
a_peFormat	an eMTK_YUV_IMAGE_FORMAT which specifies the color format of the image. The eMTK_YUV_IMAGE_FORMAT data type is an enumeration which defines the following values: • eMTK_YUV_IMAGE_FORMAT_MIN – YUV 4:1:1 format • eMTK_YUV_IMAGE_FORMAT_411 – eMTK_YUV_IMAGE_FORMAT_MIN • eMTK_YUV_IMAGE_FORMAT_420 – YUV 4:2:0 format • eMTK_YUV_IMAGE_FORMAT_422 – YUV 4:2:2 format • eMTK_YUV_IMAGE_FORMAT_444 – YUV 4:4:4 format ***Note:*** Only YUV 4:2:0 format is currently supported.

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

■ Cautions

Only YUV 4:2:0 format is currently supported.

■ Errors

The function will fail if an invalid handle or null pointer is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **mtk_GetYUVImageFormat( )**
- **mtk_CreateImageTemplate( )**

mtk_SetYUVImageSize( )

Name: MTK_RETURN mtk_SetYUVImageSize (a_hYUVImage, a_unWidth, a_unHeight)

Inputs:

MTK_HANDLE a_hYUVImage	• handle to a YUV image template
unsigned int a_unWidth	• width value
unsigned int a_unHeight	• height value

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: mtklib.h

Category: Media Toolkit

Mode: synchronous

Description

The **mtk_SetYUVImageSize( )** function sets the dimensions for a YUV image template.

Parameter	Description
a_hYUVImage	handle to a YUV image template returned from **mtk_CreateImageTemplate( )** for which to set the current image size
a_unWidth	the YUV image template width
a_unHeight	the YUV image template height

The function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

The function will fail if an invalid handle or null pointer is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information. For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **mtk_CreateImageTemplate( )**
- **mtk_GetYUVImageSize( )**

ob_CreateImageOverlayTemplate( )

Name: OB_OVERLAY_HANDLE ob_CreateImageOverlayTemplate (a_hImage)

Inputs: MTK_HANDLE a_hImage
- handle to image

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_CreateImageOverlayTemplate( )** function creates a media image overlay template. This function returns a handle to a media image overlay template that can then be applied to a supported streaming device using **sm_AddOverlays( )**.

Set and get the attributes of an overlay template using various ob_SetOverlay*( ) and ob_GetOverlay*( ) functions. Changes to a given overlay template through various ob_SetOverlay*( ) functions do not affect those added, or re-added, to a device. In other words, the device keeps a snapshot of the overlay from the time it was added.

Call **ob_DestroyOverlayTemplate( )** when the application is done with the overlay template.

Parameter	Description
a_hImage	handle returned from **mtk_CreateMediaFileTemplate( )** or **mtk_CreateBitmapTemplate( )**

The function returns an OB_OVERLAY_HANDLE if successful; otherwise, it returns MTK_ERROR.

Cautions

Be sure to call **ob_DestroyOverlayTemplate( )** when the application is done with the overlay template.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- ob_SetOverlay*( ) functions
- ob_GetOverlay*( ) functions
- **mtk_CreateBitmapTemplate( )**
- **mtk_CreateMediaFileTemplate( )**
- **ob_DestroyOverlayTemplate( )**

ob_DestroyOverlayTemplate( )

Name: MTK_RETURN ob_DestroyOverlayTemplate (a_hOverlay)

Inputs: OB_OVERLAY_HANDLE a_hOverlay
- handle to an overlay template to be destroyed

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_DestroyOverlayTemplate( )** function destroys an overlay template.

This function releases resources directly associated with the specified handle (a_hOverlay) that were allocated by a call to **ob_CreateImageOverlayTemplate( )**. After this function is called, the associated handle is no longer valid for use with any other library functionality.

Parameter	Description
a_hOverlay	handle to an overlay template to be destroyed

Cautions

- After this function call, the associated handle is no longer valid for use with any other library functionality.
- Do not call this function on handles returned in event data from a call to **sm_AddOverlays( )**.

Errors

This function will fail if an invalid handle is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **ob_CreateImageOverlayTemplate( )**

ob_GetOverlayBoundingFrame( )

Name: MTK_RETURN ob_GetOverlayBoundingFrame (a_hOverlay, a_phFrame)

Inputs: OB_OVERLAY_HANDLE a_hOverlay — handle of an overlay template

MTK_FRAME_HANDLE * a_phFrame — pointer to an MTK_FRAME_HANDLE

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_GetOverlayBoundingFrame( )** function gets a handle to the bounding frame snapshot for an overlay template. Specify a valid handle returned from **ob_CreateImageOverlayTemplate( )** or **sm_AddOverlays( )**.

Parameter	Description
a_hOverlay	handle of the overlay template for which to get the bounding frame
a_phFrame	pointer to an MTK_FRAME_HANDLE to be filled with the handle of the current bounding frame for the specified overlay. This parameter is set to -1 if the bounding frame has not been set for the given overlay.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **ob_SetOverlayBoundingFrame( )**
- **ob_CreateImageOverlayTemplate( )**

ob_GetOverlayDuration( )

Name: MTK_RETURN ob_GetOverlayDuration (a_hOverlay, a_punDuration)

Inputs: OB_OVERLAY_HANDLE a_hOverlay
- handle to an overlay template

unsigned int * a_punDuration
- pointer to duration of play

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_GetOverlayDuration( )** function gets the length of time, in milliseconds, that an overlay plays over any stream on a device. Specify a valid handle to an overlay returned from **ob_CreateImageOverlayTemplate( )** or **sm_AddOverlays( )**.

Parameter	Description
a_hOverlay	handle of the overlay template for which to get the duration of play
a_punDuration	pointer to duration of play

Cautions

None.

Errors

This function will fail if an invalid handle or NULL pointer is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **ob_SetOverlayDuration( )**
- **ob_CreateImageOverlayTemplate( )**

ob_SetOverlayBoundingFrame( )

Name: MTK_FRAME_HANDLE ob_SetOverlayBoundingFrame (a_hOverlay, a_hFrame)

Inputs: OB_OVERLAY_HANDLE a_hOverlay — • handle of an overlay template

MTK_FRAME_HANDLE a_hFrame — • handle to a frame template

Returns: overlay handle if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_SetOverlayBoundingFrame( )** function sets the bounding frame for an overlay template. The bounding frame is the space and location in which the overlay is sized to fit on the screen. A snapshot of the frame is taken and stored to be referenced by the handle returned by this function. This handle to the shapshot can be used to change the properties of the frame associated with the overlay without changing the original template.

Specify a valid handle returned from **ob_CreateImageOverlayTemplate( )** or **sm_AddOverlays( )**.

After setting the position and size of the intended frame template, you can apply it to an overlay using **ob_SetOverlayBoundingFrame( )**.

Parameter	Description
a_hOverlay	handle of the overlay template for which to set the bounding frame
a_hFrame	a handle to a frame template created by a call to **mtk_CreateFrameTemplate( )**

If successful, this function returns a handle to a snapshot frame template containing the parameters of the frame template; otherwise, this function returns MTK_ERROR.

Cautions

Do not call **mtk_DestroyFrameTemplate( )** on the handle returned by this function.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **mtk_SetFramePosition( )**
- **mtk_SetFrameSize( )**
- **ob_GetOverlayBoundingFrame( )**
- **ob_CreateImageOverlayTemplate( )**

ob_GetOverlayFillStyle( )

Name: MTK_RETURN ob_GetOverlayFillStyle (a_hOverlay, a_peFillStyle)

Inputs: OB_OVERLAY_HANDLE a_hOverlay — • handle of an overlay template

eOB_FILL_STYLE * a_peFillStyle — • fill style

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_GetOverlayFillStyle( )** function gets the current overlay fill style for an overlay template. The fill style is the manner in which the bounding frame is filled by the media set in the overlay.

Specify a valid handle returned from **ob_CreateImageOverlayTemplate( )** or **sm_AddOverlays( )**.

Parameter	Description
a_hOverlay	handle of the overlay template for which to get the fill style
a_peFillStyle	pointer to be filled with the fill style set in the overlay

This function returns MTK_SUCCESS if successful; otherwise, it returns MTK_ERROR.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, “Error Codes”.

Example

For example code, see Section 6.2, “Stream Manipulation Media Toolkit Example Code”, on page 125.

See Also

- **ob_SetOverlayFillStyle( )**
- **ob_CreateImageOverlayTemplate( )**

ob_SetOverlayDuration( )

Name: MTK_RETURN ob_SetOverlayDuration (a_hOverlay, a_unDuration)

Inputs: OB_OVERLAY_HANDLE a_hOverlay — handle to an overlay template

unsigned int a_unDuration — duration of the play

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_SetOverlayDuration( )** function sets the length of time, in milliseconds, for an overlay to be played from the beginning of any stream on the device with which it is associated. Specify a valid handle to an overlay returned from **ob_CreateImageOverlayTemplate( )** or **sm_AddOverlays( )**.

Parameter	Description
a_hOverlay	handle of the overlay template for which to set the duration of the play
a_unDuration	length of time, in milliseconds, that the overlay is to be played on the associated stream or MTK_INFINITE for the overlay to last the duration of the stream

Cautions

None.

Errors

This function will fail if an invalid handle is specified. If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **ob_GetOverlayDuration( )**
- **ob_CreateImageOverlayTemplate( )**

ob_SetOverlayFillStyle( )

Name: MTK_RETURN ob_SetOverlayFillStyle (a_hOverlay, a_eFillStyle)

Inputs: OB_OVERLAY_HANDLE a_hOverlay — • handle of an overlay template

eOB_FILL_STYLE a_eFillStyle — • fill style

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: ob_mtklib.h

Category: Overlay Builder

Mode: synchronous

Description

The **ob_SetOverlayFillStyle( )** function sets the overlay fill style for an overlay template. The fill style is the manner in which the bounding frame is filled by the media set in the overlay.

Specify a valid handle returned from **ob_CreateImageOverlayTemplate( )** or **sm_AddOverlays( )**.

Parameter	Description
a_hOverlay	handle of the overlay template for which to set the fill style
a_eFillStyle	the fill style to be used for the overlay. The eOB_FILL_STYLE data type is an enumeration which defines the following values: ***Note:*** Currently only eOB_FILL_STYLE_RESIZE_TO_FIT is supported. • eOB_FILL_STYLE_RESIZE_TO_FIT – Overlay is resized to fit the frame. Image is resized in both x and y directions to fill the entire bounding frame; this may cause distortion in the original image. • eOB_FILL_STYLE_NO_CHANGE_IN_SIZE – Overlay is presented as is with no resizing. • eOB_FILL_STYLE_MAINTAIN_ASPECT_RATIO – Overlay is stretched to maintain aspect ratio of original; resizing occurs until one side of the overlay hits either side of the frame.

This function returns MTK_SUCCESS if successful; otherwise, it function returns MTK_ERROR.

Cautions

None.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about error codes, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **ob_GetOverlayFillStyle( )**
- **ob_CreateImageOverlayTemplate( )**

sm_AddOverlays( )

Name: MTK_RETURN sm_AddOverlays (a_hDev, a_pOverlayList, a_pUserInfo)

Inputs:

SRL_DEVICE_HANDLE a_hDev	• device handle
PSM_ADD_OVERLAY_LIST a_pOverlayList	• pointer to add overlay list
void * a_pUserInfo	• pointer to user-defined data

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: sm_mtklib.h

Category: Stream Manipulation

Mode: asynchronous

Description

The **sm_AddOverlays( )** function adds one or more overlays to a device. The overlays are placed over any media stream generated by that device. A maximum of two overlays per device is supported at any given time.

Overlays are defined using templates created with **ob_CreateImageOverlayTemplate( )**. If any of the overlays associated with the given overlay handles in **a_pOverlayList** fails to be added, the function fails and no overlays are added.

The overlay template handle in each SM_ADD_OVERLAY structure is mapped to a snapshot of the current state of the overlay template. This mapping is returned in the event data of this function.

The **sm_AddOverlays( )** function can be called while the device is streaming data.

Parameter	Description
a_hDev	device handle on which to add the overlay
a_pOverlayList	pointer to an add overlay list in the SM_ADD_OVERLAY_LIST structure
a_pUserInfo	pointer to user-defined data. If none, set to NULL.

This function returns MTK_SUCCESS if successful and MTK_ERROR for failure.

Overlays are defined using templates created with **ob_CreateImageOverlayTemplate( )**. For each overlay being added to a device, a snapshot of the parameters for that overlay is used to create the overlay on the device. The snapshot overlay handle is returned in the completion data for the successful add operation, allowing the user to map from the template to the snapshot for that device.

If a snapshot handle is used for an add operation a second time on the same device by setting SM_ADD_OVERLAY hOverlay to the snapshot handle, the current parameters for the snapshot

overlay are sent to the device and the overlay is reset on that device. This is considered a modification to the overlay on that device. In this case, both the snapshot and template overlay handles in the event data will be the snapshot overlay handle.

If a snapshot handle is used for an add operation on a different device, that snapshot overlay is considered a template on the second device. The current parameters for the snapshot overlay are sent to the device and the overlay is added on the second device.

In all cases, use the snapshot overlay handle returned in SM_ADD_OVERLAY_RESULT to remove the overlay from the device by making a call to **sm_RemoveOverlays( )** for the given device.

Changes to a given overlay template through the ob_Set*( ) functions do not affect those overlays already added to a device until **sm_AddOverlays( )** is called again using the snapshot handle associated with the given overlay. In other words, the snapshot of the overlay is not updated in the firmware until an additional call to **sm_AddOverlays( )** is made, presenting the snapshot handle as the template handle in the SM_ADD_OVERLAY structure.

Events

If the function returns MTK_SUCCESS, the following events may be generated.

SMEV_ADD_OVERLAY

Termination event reported on successful completion of adding overlay(s).

Event Data: The event data is an SM_ADD_OVERLAY_RESULT_LIST structure. The SM_ADD_OVERLAY_RESULT_LIST structure contains a list of SM_ADD_OVERLAY_RESULT structure instances. Each result structure instance contains a mapping from one of the template overlay handles in the original SM_ADD_OVERLAY_LIST to the handle of the snapshot overlay. This snapshot overlay handle is used in the call to **sm_RemoveOverlays( )** to remove this instance of the overlay for the given device.

SMEV_ADD_OVERLAY_FAIL

Termination event reported upon encountering an error while adding overlay(s).

Event Data: An SM_ADD_OVERLAY_RESULT_LIST structure.

When there is an add failure, all overlays referenced in **a_pOverlayList** are considered invalidated. If any of the overlays referenced had been added previously, each overlay is removed from the device and from the system. Overlay templates created with ob_Create*OverlayTemplate( ) are not affected. The event data is an SM_ADD_OVERLAY_RESULT_LIST structure containing an SM_ADD_OVERLAY_RESULT for each overlay referenced. The hOverlayTemplate in each SM_ADD_OVERLAY_RESULT is set to the handle of the affected overlay. The result data member of each SM_ADD_OVERLAY_RESULT is set to eSM_ADD_OVERLAY_RESULT_FAIL.

SMEV_ERROR

Unsolicited event reported upon encountering an unexpected error.

Cautions

- You must set the bounding frame for an overlay template before adding the overlay to a device or the add overlay operation will fail. Call **ob_SetOverlayBoundingFrame( )** to set the bounding frame.
- Do not call **ob_DestroyOverlayTemplate( )** on the handles returned in the add overlay completion event data.
- This function fails if it is called on a device connected using **dev_Connect( )** or **dev_PortConnect( )** in native mode (DMFL_TRANSCODE_NATIVE). To use this function, transcoding mode must be set (DMFL_TRANSCODE_ON).
- The **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** functions are not supported for this function.

Errors

If this function fails, call **mtk_GetErrorInfo( )** for error information.

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **sm_RemoveOverlays( )**
- **sm_RemoveAllOverlays( )**

sm_RemoveAllOverlays( )

Name: MTK_RETURN sm_RemoveAllOverlays (a_hDev, a_pUserInfo)

Inputs: SRL_DEVICE_HANDLE a_hDev
- device handle

void * a_pUserInfo
- pointer to user-defined data

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: sm_mtklib.h

Category: Stream Manipulation

Mode: asynchronous

Description

The **sm_RemoveAllOverlays( )** function stops and removes all overlays previously added to a device. This function can be called while the device is streaming data.

Parameter	Description
a_hDev	device handle from which to remove all overlays
a_pUserInfo	pointer to user-defined data. If none, set to NULL.

This function returns MTK_SUCCESS if successful and MTK_ERROR for failure.

Events

If the function returns MTK_SUCCESS, the following events may be generated.

SMEV_REMOVE_ALL_OVERLAYS
Termination event reported on successful completion of removing all overlays.
Event Data: None.

SMEV_REMOVE_ALL_OVERLAYS_FAIL
Termination event reported upon encountering an error while removing overlays.
Event Data: None.

SMEV_ERROR
Unsolicited event reported upon encountering an unexpected error.
Event Data: None.

Cautions

- The **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** functions are not supported for this function.

Errors

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **sm_AddOverlays( )**
- **sm_RemoveOverlays( )**

sm_RemoveOverlays( )

Name: MTK_RETURN sm_RemoveOverlays (a_hDev, a_pOverlayList, a_pUserInfo)

Inputs:

SRL_DEVICE_HANDLE a_hDev	• device handle
PSM_ADD_OVERLAY_LIST a_pOverlayList	• pointer to overlay list
void * a_pUserInfo	• pointer to user-defined data

Returns: MTK_SUCCESS if successful
MTK_ERROR on failure

Includes: sm_mtklib.h

Category: Stream Manipulation

Mode: asynchronous

Description

The **sm_RemoveOverlays( )** function stops and removes one or more overlays previously added to a device. This function can be called while the device is streaming data.

Parameter	Description
a_hDev	device handle from which to remove one or more overlays.
a_pOverlayList	pointer to an overlay list in the SM_ADD_OVERLAY_LIST structure. Valid overlay handles are the snapshot overlay handles returned in the add completion event data for **sm_AddOverlays( )**.
a_pUserInfo	pointer to user-defined data. If none, set to NULL.

This function returns MTK_SUCCESS if successful and MTK_ERROR for failure.

Events

If the function returns MTK_SUCCESS, the following events may be generated.

SMEV_REMOVE_OVERLAY
Termination event reported on successful completion of removing overlay(s).
Event Data: The event data is an SM_REMOVE_OVERLAY_LIST structure.

SMEV_REMOVE_OVERLAY_FAIL
Termination event reported upon encountering an error while removing overlay(s).
Event Data: None.

SMEV_ERROR
Unsolicited event reported upon encountering an unexpected error.
Event Data: None.

Cautions

- The **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** functions are not supported for this function.
- Do not call this function on overlay handles returned from **ob_CreateImageOverlayTemplate( )**.

Errors

For more information about errors, see Chapter 5, "Error Codes".

Example

For example code, see Section 6.2, "Stream Manipulation Media Toolkit Example Code", on page 125.

See Also

- **sm_AddOverlays( )**
- **sm_RemoveAllOverlays( )**

Events 3

This chapter provides information about the events that may be returned by the Dialogic® Media Toolkit API software.

An event indicates that a specific activity has occurred on a channel. The host library reports channel activity to the application program in the form of events, which allows the program to identify and respond to a specific occurrence on a channel. Events provide feedback on the progress and completion of functions and indicate the occurrence of other channel activities. Dialogic® Media Toolkit API library events are defined in the *sm_mtkevts.h* header file.

A termination event is returned after the completion of a function call operating in asynchronous mode. The Dialogic® Media Toolkit API library provides a pair of termination events for a function, to indicate successful completion or failure. A termination event is only generated in the process that called the function.

Use **sr_waitevt( )**, **sr_enbhdlr( )** or other SRL functions to collect an event code, depending on the programming model in use. For more information, see the *Dialogic® Standard Runtime Library API Library Reference*.

The following events, listed in alphabetical order, may be returned by the Dialogic® Media Toolkit API software.

SMEV_ADD_OVERLAY
Termination event for **sm_AddOverlays( )**. Specified overlay(s) were added successfully.

SMEV_ADD_OVERLAY_FAIL
Termination event for **sm_AddOverlays( )**. Add overlay operation failed.

SMEV_ERROR
Unsolicited general error event. Returned when an unexpected error occurs.

SMEV_REMOVE_ALL_OVERLAYS
Termination event for **sm_RemoveAllOverlays( )**. All overlays were removed successfully .

SMEV_REMOVE_ALL_OVERLAYS_FAIL
Termination event for **sm_RemoveAllOverlays( )**. Remove all overlays operation failed.

SMEV_REMOVE_OVERLAY
Termination event for **sm_RemoveOverlays( )**. Specified overlay(s) were removed successfully.

SMEV_REMOVE_OVERLAY_FAIL
Termination event for **sm_RemoveOverlays( )**. Remove overlay(s) operation failed.

Data Structures 4

This chapter provides an alphabetical reference to the data structures used by the Dialogic® Media Toolkit API software. The following data structures are described:

MP_FILE_INFO_DESC

```
typedef struct tagMP_FILE_INFO_DESC
{
      unsigned int unVersion;                     //Input - Structure version
      unsigned short unsigned char *fileInfoSize; //Input -  indicate the size of
                                                          File Info block
      unsigned short unsigned char *pFileInfo;    //Input - Pointer to File Info block - set by
                                                    the application
      unsigned short  fileType;                  //Output - Type of file
      char format[8];                            //Output - File format standard
      unsigned long version;                     //Output - Format version
      unsigned long flags;                       //Output - see #MP_DESC_FLAG_INSUFF_SPACE
      unsigned long   NbVideoTracks;             //Output - Number of video tracks present
} MP_FILE_INFO_DESC;
```

Description

This structure contains the descriptor for the multimedia file.

Field Descriptions

The fields of this data structure are described as follows:

unVersion
: specifies the version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the MP_FILE_INFO_DESC_VERSION macro for setting the current version.

fileInfoSize
: used as both an input and an output. Set by the application to indicate the size of File Info block being provided (see pFileInfo). Reset by **mp_GetFileInfo( )** to indicate the actual size of block used.

pFileInfo
: used as both an input and an output. Pointer to the file information block. The application is responsible for allocating a block of memory for the file information block and providing to the **mp_GetFileInfo( )**. It can use **mp_GetFileInfoSize( )** to determine the minimum size necessary, but this is optional. However, if the size provided is not large enough to retrieve all of the file information, the file information may be truncated and the error condition is noted with the flags parameter inside this structure. For ISO-based multimedia files like 3GP, 20 kbytes is normally enough to return all of the file information.

Note: The remaining field descriptions for this data structure are not provided because the fields are for output only.

MP_MMFILE

```
typedef struct tagMP_MMFILE
{
        unsigned int    unVersion;          //Input - Structure version
        unsigned        state;
        unsigned        openMode;
        unsigned        fileType;
        unsigned        verboseLevel;
        void            *mp;
        int             streamCount;
        void            *pMMStream;
        int             track_ids_set;
        unsigned long   video_track_id;
        unsigned long   audio_track_id;

} MP_MMFILE;
```

Description

This structure is used by the Media Parsing functions to access a multimedia file. This structure is created by the application and is filled-in by calling **mp_OpenFile( )**. It is then passed as an input to the other Media Parsing functions. There is only a single field that is an input and needs to be set by the application, the unVersion field.

Field Descriptions

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

unVersion
specifies the structure version. Use the predefined macro, MP_MMFILE_VERSION for setting this field.

Note: Field descriptions for this data structure are not provided because the fields are internal and not used by the application.

MTK_BITMAP_DATA

```
typedef struct
{
   unsigned int     unVersion;  ///< structure version
   unsigned char *  pucData;    ///< bitmap data
   unsigned int     unLength;   ///< bitmap data length

} MTK_BITMAP_DATA, *PMTK_BITMAP_DATA;
typedef const MTK_BITMAP_DATA * CPMTK_BITMAP_DATA;
```

Description

The MTK_BITMAP_DATA structure stores bitmap data settings. This structure is used by the **mtk_SetBitmapData( )** and **mtk_GetBitmapData( )** functions.

The INIT_MTK_BITMAP_DATA inline function is provided to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

pucData
: bitmap data

unLength
: bitmap data length

MTK_ERROR_INFO

```
typedef struct
{
   unsigned int unVersion;          ///< Structure version
   unsigned int unErrorCode;        ///< Error code
   const char * szErrorString;      ///< Error string
   const char * szAdditionalInfo;   ///< Additional error information string

} MTK_ERROR_INFO, *PMTK_ERROR_INFO;
typedef const MTK_ERROR_INFO* CPMTK_ERROR_INFO;
```

Description

The MTK_ERROR_INFO structure provides error information when an API function call fails. This structure is used by the **mtk_GetErrorInfo( )** function.

Use the INIT_MTK_ERROR_INFO inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

unErrorCode
: error code

szErrorString
: error message

szAdditionalInfo
: additional error information

MTK_RECT

```
typedef struct
{
   unsigned int   unVersion;  ///< Structure version
   float          x;          ///< Horizontal offset
   float          y;          ///< Vertical offset
   float          width;      ///< Rectangle width
   float          height;     ///< Rectangle height

} MTK_RECT, *PMTK_RECT;
typedef const MTK_RECT * CPMTK_RECT;
```

Description

The MTK_RECT structure stores the position and dimensions of a rectangle. The rectangle is the space in which an object, such as a layout, is placed on the screen. The x/y position is relative to the upper, left-hand corner of the screen. The dimensions are expressed in percentages.

This structure is used by the **lb_SetRect( )** and **lb_GetRect( )** functions.

Use the INIT_MTK_RECT inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

x
: horizontal offset. For example, for a layout type of four equal regions, the top left region would be defined as 0 percent offset; the top right would be 50 percent offset; the bottom right would be 50 percent offset; the bottom left would be 0 percent offset.

y
: vertical offset. For example, for a layout type of four equal regions, the top left region would be defined as 0 percent offset; the top right would be 0 percent offset; the bottom right would be 50 percent offset; the bottom left would be 50 percent offset.

width
: width of rectangle. For example, for a layout type of four equal regions, the top left region would be defined as 50 percent offset; the top right would be 50 percent offset; the bottom right would be 50 percent offset; the bottom left would be 50 percent offset.

height
: height of rectangle. For example, for a layout type of four equal regions, the top left region would be defined as 50 percent offset; the top right would be 50 percent offset; the bottom right would be 50 percent offset; the bottom left would be 50 percent offset.

MTK_STRING

```
typedef struct
{
   unsigned int unVersion;      ///< structure version
   char *       szString;       ///< string
   unsigned int unLength;       ///< string length

} MTK_STRING, *PMTK_STRING;
typedef const MTK_STRING * CPMTK_STRING;
```

Description

The MTK_STRING structure stores the media file name. This structure is used by the **mtk_SetMediaFileName( )** and **mtk_GetMediaFileName( )** functions.

Use the INIT_MTK_STRING inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

szString
: string

unLength
: length of string

SM_ADD_OVERLAY

```
typedef struct
{
   unsigned int unVersion;              /*! structure version */
   eSM_OVERLAY_DIRECTION eDirection;    /*! direction of overlay */
   OB_OVERLAY_HANDLE hOverlay;          /*! overlay handle */

} SM_ADD_OVERLAY, *PSM_ADD_OVERLAY;
typedef const PSM_ADD_OVERLAY CPSM_ADD_OVERLAY;
```

Description

The SM_ADD_OVERLAY structure contains add overlay information. This structure is used by the **sm_AddOverlays( )** function.

Use the INIT_SM_ADD_OVERLAY inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eDirection
: direction in which the overlay is presented. The eSM_OVERLAY_DIRECTION data type is an enumeration which defines the following values:
  - eSM_OVERLAY_DIRECTION_DEVICE – The overlay is placed on the data stream between the device in question and any other device with which it is connected.
  - eSM_OVERLAY_DIRECTION_NETWORK – The overlay is placed on the data stream between the device in question and the external network. For a media device that does not normally have a network, such as a multimedia device, the external network is considered the sink associated with that device such as a recorded file.

hOverlay
: overlay handle obtained from **ob_CreateImageOverlayTemplate( )** or snapshot handle that is returned in SM_ADD_OVERLAY_RESULT structure after a call to **sm_AddOverlays( )**.

SM_ADD_OVERLAY_LIST

```
typedef struct
{
   unsigned int unVersion;                      /*! structure version */
   unsigned int unCount;                        /*! number of valid overlays in addOverlays */
   SM_ADD_OVERLAY addOverlays[SM_MAX_OVERLAYS];  /*! add overlay list of overlays */

} SM_ADD_OVERLAY_LIST, *PSM_ADD_OVERLAY_LIST;
typedef const PSM_ADD_OVERLAY_LIST CPSM_ADD_OVERLAY_LIST;
```

Description

The SM_ADD_OVERLAY_LIST structure contains information about add overlay lists. This structure is used by the **sm_AddOverlays( )**.

Use the INIT_SM_ADD_OVERLAY_LIST inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

unCount
: number of valid overlays contained in addOverlays

addOverlays
: add overlay list of overlays in SM_ADD_OVERLAY structure

SM_ADD_OVERLAY_RESULT

```
typedef struct
{
   unsigned int unVersion;                  /*! structure version */
   OB_OVERLAY_HANDLE hOverlayTemplate;      /*! overlay template handle */
   OB_OVERLAY_HANDLE hOverlaySnapshot;      /*! overlay snapshot handle */
   eSM_ADD_OVERLAY_RESULT result;           /*! add result code */

} SM_ADD_OVERLAY_RESULT, *PSM_ADD_OVERLAY_RESULT;
typedef const PSM_ADD_OVERLAY_RESULT CPSM_ADD_OVERLAY_RESULT;
```

Description

The SM_ADD_OVERLAY_RESULT structure stores the results of the add overlay operation. This structure is used by the **sm_AddOverlays( )** function.

Use the INIT_SM_ADD_OVERLAY_RESULT inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

hOverlayTemplate
: handle of overlay template

hOverlaySnapshot
: handle of snapshot overlay

result
: result code. The eSM_ADD_OVERLAY_RESULT data type is an enumeration which defines the following values:
  - eSM_ADD_OVERLAY_RESULT_ADD – Create a new overlay for the device; overlay identifier is unique for the given device
  - eSM_ADD_OVERLAY_RESULT_MODIFY – Modify existing overlay for the device; identifier is already associated with an overlay for the given device.
  - eSM_ADD_FAIL – Add overlay failed.

SM_ADD_OVERLAY_RESULT_LIST

```
typedef struct
{
   unsigned int unVersion;     /*! structure version */
   unsigned int unCount;       /*! number of add overlay results contained in overlayResults */
   SM_ADD_OVERLAY_RESULT overlayResults[SM_MAX_OVERLAYS];  /*! array of overlay results */
} SM_ADD_OVERLAY_RESULT_LIST, *PSM_ADD_OVERLAY_RESULT_LIST;

typedef const PSM_ADD_OVERLAY_RESULT_LIST CPSM_ADD_OVERLAY_RESULT_LIST;
```

Description

The SM_ADD_OVERLAY_RESULT_LIST structure contains a list of add overlay results. This structure is used by the **sm_AddOverlays( )** function.

Use the INIT_SM_ADD_OVERLAY_RESULT_LIST inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

unCount
: number of add overlay results contained in overlayResults

overlayResults
: array of overlay results in SM_ADD_OVERLAY_RESULT structure

SM_REMOVE_OVERLAY_LIST

```
typedef struct
{
   unsigned int unVersion;      /*! structure version */
   unsigned int unCount;        /*! number of overlay handles contained in OverlayHandles */
   OB_OVERLAY_HANDLE  overlayHandles[SM_MAX_OVERLAYS];  /*! array of overlay handles */
} SM_REMOVE_OVERLAY_LIST, *PSM_REMOVE_OVERLAY_LIST;

typedef const PSM_REMOVE_OVERLAY_LIST CPSM_REMOVE_OVERLAY_LIST;
```

Description

The SM_REMOVE_OVERLAY_LIST structure contains remove overlay list data. This structure is used by the **sm_RemoveOverlays( )** function.

Use the INIT_SM_REMOVE_OVERLAY_LIST inline function to initialize the structure.

Field Descriptions

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

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

unCount
: number of valid overlay handles contained in overlayHandles

overlayHandles
: array of overlay handles. Valid overlay handles are the snapshot overlay handles returned in the add completion event data for **sm_AddOverlays( )**.

Error Codes 5

This chapter describes the error codes used by the Dialogic® Media Toolkit API software.

Error codes are defined in *mtkerrs.h*, with the exception of Media Parsing API errors. These errors are defined in *mp_mtklib.h* and in Section 5.1, "Media Parsing API Errors", on page 114.

Dialogic® Media Toolkit API library functions return a value that indicates the success or failure of a function call. Success is indicated by MTK_SUCCESS, and failure is indicated by MTK_ERROR. If a library function returns MTK_ERROR to indicate failure, use **mtk_GetErrorInfo( )** to obtain the reason for the error.

If an error occurs during execution of an asynchronous function, an error event, preceded by "MTKEV_" is sent to the application. No change of state is triggered by this event. Upon receiving the MTKEV_ERROR event, the application can retrieve the reason for the failure using **mtk_GetErrorInfo( )**.

The error codes used by the Dialogic® Media Toolkit API software are described as follows:

EMTK_FIRMWARE
: firmware error

EMTK_INVALID_ATTR
: invalid device attribute

EMTK_INVALID_DEVICE
: invalid device

EMTK_INVALID_EVENT
: invalid device event

EMTK_INVALID_HANDLE
: invalid device handle

EMTK_INVALID_NAME
: invalid device name

EMTK_INVALID_PARM
: invalid parameter

EMTK_INVALID_STATE
: invalid device state for requested operation

EMTK_LIBRARY
: library error

EMTK_MEMORY_ALLOC
: memory allocation error

EMTK_NOERROR
: no error

EMTK_SUBSYSTEM
internal subsystem error

EMTK_SYSTEM
system error

EMTK_UNSUPPORTED_API
API not currently supported

EMTK_UNSUPPORTED_FUNC
requested functionality not supported

EMTK_UNSUPPORTED_TECH
technology not currently supported

5.1 Media Parsing API Errors

The error codes used by the Media Parsing API are described as follows:

MPERR_ FILE_ACCESS_DENIED
access to file denied

MPERR_FILE_FORMAT
file format error , access denied

MPERR_FILE_OPEN_FAILED
file not found or MP_OpenFile( ) failed

MPERR_INCOMP_ACCESS_MODE
file not opened for read

MPERR_ INTERNAL_RESOURCE
internal resource error (e.g., memory allocation)

MPERR_ INVALID_FILE
pMMFile invalid

MPERR_ INVALID_FILE_TYPE
file type not supported

MPERR_ INVALID_POINTER
Null pFileInfoDesc, pMMFile ,or pFileName

MPERR_INVALID_STRUCT_VERSION
invalid version of pFileInfoDesc

Note: Do not call the **mtk_GetErrorInfo( )** function to retrieve error information for Media Parsing functions.

Supplementary Reference Information 6

This chapter provides supplementary reference information about the Dialogic® Media Toolkit API library. The following topics are covered:

6.1 Layout Builder Media Toolkit Example Code

Written in the C++ programming language, the example code in Figure 5 illustrates the layout builder functionality. The code exercises several functions and data structures that are part of the Dialogic® Media Toolkit API library, including media toolkit and layout builder functions.

The example code is intended to illustrate how the Dialogic® Media Toolkit API functions are used in a simple application. It is not intended to be used in a production environment.

The output from the example code is provided in Figure 6.

Figure 5. Layout Builder Media Toolkit Example Code

```
#include <lb_mtklib.h>
#include <iostream>

#ifdef WIN32
#else
#include <unistd.h>
#endif

using namespace std;

#define MAX_REGIONS 10

void DumpErrorInfo();
char * GetLayoutTypeString(eLB_LAYOUT_TYPE a_eType);

/**
 * @fn main
 */
int main(int nArgCount, char *pArgList[])
{
   cout << "Layout Builder Media Toolkit Sample Code" << endl;
   cout << "========================================" << endl << endl;

   LB_FRAME_HANDLE LayoutHandleList[3];
   MTK_RETURN MTKResult;

/*****************************************************************************
 * CREATE AN LB_LAYOUT_TYPE_1_1 VIDEO LAYOUT
 *****************************************************************************/
```

```
   LayoutHandleList[0] = lb_CreateLayoutTemplate(eLB_LAYOUT_TYPE_1_1);
   if (LayoutHandleList[0] == MTK_ERROR)
   {
      cout << "lb_CreateLayoutTemplate for 1_1 layout failed !!" << endl;
             DumpErrorInfo();
      return 0;
   }
   else
   {
      cout << "Create 1_1 Video Layout Successfully... Handle: " << LayoutHandleList[0] << endl;
   }

   /*************************************************************************
    * CREATE AN LB_LAYOUT_TYPE_4_1 VIDEO LAYOUT
    *************************************************************************/
   LayoutHandleList[1] = lb_CreateLayoutTemplate(eLB_LAYOUT_TYPE_4_1);
   if (LayoutHandleList[1] == MTK_ERROR)
   {
      cout << "lb_CreateLayoutTemplate for 4_1 layout failed !!" << endl;
               DumpErrorInfo();
      return 0;
   }
   else
   {
      cout << "Created 4_1 Video Layout Successfully... Handle: " << LayoutHandleList[1] <<
endl;
   }

   /*************************************************************************
    * CREATE A CUSTOM VIDEO LAYOUT
    *************************************************************************/
   LayoutHandleList[2] = lb_CreateLayoutTemplate(eLB_LAYOUT_TYPE_CUSTOM);
   if (LayoutHandleList[2] == MTK_ERROR)
   {
      cout << "lb_CreateLayout for CUSTOM layout failed !!" << endl;
      DumpErrorInfo();
      return 0;
   }
   else
   {
      cout << "Created CUSTOM Video Layout Successfully... Handle: " << LayoutHandleList[2] <<
endl;
   }

   int i;
   LB_FRAME_HANDLE LayoutHandle;
   LB_FRAME_HANDLE RegionHandle;
   eLB_LAYOUT_TYPE eLayoutTypeList[3];

   for (i = 0; i < 3; i++)
   {
      eLB_LAYOUT_TYPE eLayoutType;
      LayoutHandle = LayoutHandleList[i];
      MTK_RECT RegionRect;

      cout << endl;

      /************************************************************************
       * Get Layout Type
       ************************************************************************/
      if (lb_GetType(LayoutHandle, &eLayoutTypeList[i]) == MTK_SUCCESS)
      {
         cout << "*************************************************" << endl;
        cout << "** Performing operation on the " << GetLayoutTypeString(eLayoutTypeList[i]) <<
" layout type." << endl;
         cout << "*************************************************" << endl << endl;
      }
```

```
      else
      {
         cout << "*************************************************" << endl;
         cout << "** INVALID LAYOUT HANDLE - CANNOT PERFORM OPERATIONS" << endl;
         cout << "*************************************************" << endl << endl;
         DumpErrorInfo();
         cout << "*************************************************" << endl << endl;
      }

      /*****************************************************************************
       * GET LAYOUT ATTRIBUTES
       *****************************************************************************/

      /*****************************************************************************
       * GET LAYOUT NAME
       *****************************************************************************/
      char pszLayoutName[128];
      size_t NameSize = 128;
      if (lb_GetName(LayoutHandle, pszLayoutName, &NameSize) == MTK_SUCCESS)
      {
         cout << "Layout Name is: " << pszLayoutName << endl << endl;
      }
      else
      {
         cout << "lb_GetName failed" << endl;
         DumpErrorInfo();
      }

      INIT_MTK_RECT(&RegionRect);
      RegionRect.height = 50.00;
      RegionRect.width = 50.00;

      /*****************************************************************************
      * ADD REGION TO LAYOUT - THIS WILL FAIL FOR NON-CUSTOM TYPE LAYOUTS
      *****************************************************************************/
      if ((RegionHandle = lb_AddRegion(LayoutHandle, &RegionRect)) == MTK_ERROR)
      {
         if (eLayoutTypeList[i] == eLB_LAYOUT_TYPE_CUSTOM)
         {
            cout << "lb_AddRegion failed" << endl;
            DumpErrorInfo();
         }
         else
         {
            cout << "lb_AddRegion failed, Expected error due to layout type." << endl;
         }
      }
      else
      {
         cout << "lb_AddRegion successfull" << endl;
      }

      /*****************************************************************************
      * GET REGION LIST
      *****************************************************************************/
      LB_FRAME_HANDLE RegionList[MAX_REGIONS];
      unsigned int RegionCount = MAX_REGIONS;
      MTKResult = lb_GetRegionList(LayoutHandle, RegionList, &RegionCount);
      if (MTKResult == MTK_ERROR)
      {
         cout << "lb_GetRegionList failed !!" << endl;
      }
      else
      {
         cout << "\nlb_GetRegionList successful... LayoutHandle: " << LayoutHandle << "
RegionCount: " << RegionCount << endl;
         for (int j = 0; j < RegionCount; j++)
```

```
{
   cout << "\tRegionHandle " << (j + 1) << ": " << RegionList[j] << endl;

   /*********************************************************************
   * GETTING REGION ATTRIBUTES
   *********************************************************************/

   /*********************************************************************
   * GET REGION NAME
   *********************************************************************/
   char pszRegionName[128];
   size_t RegionNameSize = 128;
   if (lb_GetName(RegionList[j], pszRegionName, &RegionNameSize) == MTK_SUCCESS)
   {
      cout << "\t\tRegion Name: " << pszRegionName << endl;
   }
   else
   {
      cout << "\t\tlb_GetName failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * GET REGION DISPLAY MODE
   *********************************************************************/
   eLB_DISPLAY_MODE eDisplayMode;
   if (lb_GetDisplayMode(RegionList[j], &eDisplayMode) == MTK_SUCCESS)
   {
      cout << "\t\tDisplay Mode: " << eDisplayMode << endl;
   }
   else
   {
      cout << "\t\tlb_GetDisplayMode failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * GET REGION SELECTION MODE
   *********************************************************************/
   eLB_SELECTION_MODE eSelectionMode;
   if (lb_GetSelectionMode(RegionList[j], &eSelectionMode) == MTK_SUCCESS)
   {
      cout << "\t\tSelection Mode: " << eSelectionMode << endl;
   }
   else
   {
      cout << "\t\tlb_GetSelectionMode failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * GET REGION PRIORITY
   *********************************************************************/
   unsigned int unPriority;
   if (lb_GetPriority(RegionList[j], &unPriority) == MTK_SUCCESS)
   {
      cout << "\t\tPriority: " << unPriority << endl;
   }
   else
   {
      cout << "\t\tlb_GetPriority failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * GET REGION RECT
   *********************************************************************/
```

```
   MTK_RECT Rect;
   if (lb_GetRect(RegionList[j], &Rect) == MTK_SUCCESS)
   {
      cout << "\t\tRect: X:" << (float)Rect.x << " Y:" << (float)Rect.y << " W:" <<
          (float)Rect.width << " H:" << (float)Rect.height << endl;
   }
   else
   {
      cout << "\t\tlb_GetRect failed" << endl;
      DumpErrorInfo();
   }
}

cout << endl;

for (int j = 0; j < RegionCount; j++)
{
   cout << "\tRegionHandle " << (j + 1) << ": " << RegionList[j] << endl;

   /*********************************************************************
   * SETTING REGION ATTRIBUTES
   *********************************************************************/

   /*********************************************************************
   * SET REGION NAME
   *********************************************************************/
   char pszCustomRegionName[64];
   sprintf(pszCustomRegionName, "Region %d", j+1);
   if (lb_SetName(RegionList[j], pszCustomRegionName) == MTK_SUCCESS)
   {
      cout << "\t\tlb_SetName successfull" << endl;
   }
   else
   {
      cout << "\t\tlb_SetName failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * SET REGION DISPLAY MODE
   *********************************************************************/
   if (lb_SetDisplayMode(RegionList[j], eLB_DISPLAY_MODE_LIVE) == MTK_SUCCESS)
   {
      cout << "\t\tlb_SetDisplayMode successfull" << endl;
   }
   else
   {
      cout << "\t\tlb_SetDisplayMode failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * SET REGION SELECTION MODE
   *********************************************************************/
   if (lb_SetSelectionMode(RegionList[j], eLB_SELECTION_MODE_USER_SELECT) ==
      MTK_SUCCESS)
   {
      cout << "\t\tlb_SetSelectionMode successfull" << endl;
   }
   else
   {
      cout << "\t\tlb_SetSelectionMode failed" << endl;
      DumpErrorInfo();
   }

   /*********************************************************************
   * SET REGION PRIORITY
```

```
   ****************************************************************/
   if (lb_SetPriority(RegionList[j], j+1) == MTK_SUCCESS)
   {
      cout << "\t\tlb_SetPriority successfull" << endl;
   }
   else
   {
      cout << "\t\tlb_GetPriority failed" << endl;
      DumpErrorInfo();
   }

   /****************************************************************
   * SET REGION RECT. Only allowed for CUSTOM LAYOUTS.
   ****************************************************************/
   MTK_RECT Rect;
   Rect.x = 10.00;
   Rect.y = 10.00;
   Rect.width = 80.00;
   Rect.height = 80.00;
   if (lb_SetRect(RegionList[j], &Rect) == MTK_SUCCESS)
   {
      cout << "\t\tlb_SetRect successfull" << endl;
   }
   else
   {
      if (eLayoutTypeList[i] == eLB_LAYOUT_TYPE_CUSTOM)
      {
         cout << "\t\tlb_SetRect failed" << endl;
         DumpErrorInfo();
      }
      else
      {
         cout << "\t\tlb_SetRect failed, expected error due to layout type" << endl;
      }
   }
}

cout << endl;

for (int j = 0; j < RegionCount; j++)
{
   cout << "\tRegionHandle " << (j + 1) << ": " << RegionList[j] << endl;

   /****************************************************************
   * GETTING REGION ATTRIBUTES
   ****************************************************************/

   /****************************************************************
   * GET REGION NAME
   ****************************************************************/
   char pszRegionName[128];
   size_t RegionNameSize = 128;
   if (lb_GetName(RegionList[j], pszRegionName, &RegionNameSize) == MTK_SUCCESS)
   {
      cout << "\t\tRegion Name: " << pszRegionName << endl;
   }
   else
   {
      cout << "\t\tlb_GetName failed" << endl;
      DumpErrorInfo();
   }

   /****************************************************************
   * GET REGION DISPLAY MODE
   ****************************************************************/
   eLB_DISPLAY_MODE eDisplayMode;
   if (lb_GetDisplayMode(RegionList[j], &eDisplayMode) == MTK_SUCCESS)
```

```
      {
         cout << "\t\tDisplay Mode: " << eDisplayMode << endl;
      }
      else
      {
         cout << "\t\tlb_GetDisplayMode failed" << endl;
         DumpErrorInfo();
      }

      /*****************************************************************
      * GET REGION SELECTION MODE
      *****************************************************************/
      eLB_SELECTION_MODE eSelectionMode;
      if (lb_GetSelectionMode(RegionList[j], &eSelectionMode) == MTK_SUCCESS)
      {
         cout << "\t\tSelection Mode: " << eSelectionMode << endl;
      }
      else
      {
         cout << "\t\tlb_GetSelectionMode failed" << endl;
         DumpErrorInfo();
      }

      /*****************************************************************
      * GET REGION PRIORITY
      *****************************************************************/
      unsigned int unPriority;
      if (lb_GetPriority(RegionList[j], &unPriority) == MTK_SUCCESS)
      {
         cout << "\t\tPriority: " << unPriority << endl;
      }
      else
      {
         cout << "\t\tlb_GetPriority failed" << endl;
         DumpErrorInfo();
      }

      /*****************************************************************
      * GET REGION PRIORITY
      *****************************************************************/
      MTK_RECT Rect;
      if (lb_GetRect(RegionList[j], &Rect) == MTK_SUCCESS)
      {
         cout << "\t\tRect: X:" << (float)Rect.x << " Y:" << (float)Rect.y << " W:"
              << (float)Rect.width << " H:" << (float)Rect.height << endl;
      }
      else
      {
         cout << "\t\tlb_GetRect failed" << endl;
         DumpErrorInfo();
      }
   }
}

/*****************************************************************
 * REMOVE REGION - THIS WILL FAIL FOR NON-CUSTOM TYPE LAYOUTS
 *****************************************************************/
if (lb_RemoveRegion(LayoutHandle, RegionHandle) == MTK_ERROR)
{
   if (eLayoutTypeList[i] != eLB_LAYOUT_TYPE_CUSTOM)
   {
      cout << "lb_RemoveRegion failed! Expected error due to layout type" << endl;
   }
   else
   {
      cout << "lb_RemoveRegion failed!" << endl;
      DumpErrorInfo();
```

```
         }
      }
      else
      {
         cout << "lb_RemoveRegion successfull" << endl;
      }

      cout << endl;
   }

   /*****************************************************************************
    * DESTROY ALL LAYOUTS
    *****************************************************************************/
   for (int i = 0; i < 3; i++)
   {
      if (MTKResult = lb_DestroyLayoutTemplate(LayoutHandleList[i]) == MTK_ERROR)
      {
         cout << "lb_DestroyLayoutTemplate failed !!" << endl;
         DumpErrorInfo();
      }
      else
      {
         cout << "lb_DestroyLayoutTemplate successfull... Handle: " << LayoutHandleList[i] <<
endl;
      }
   }

   return 0;
}

/**
 * @fn DumpErrorInfo
 */
void DumpErrorInfo()
{
   MTK_ERROR_INFO ErrorInfo;
   if (mtk_GetErrorInfo(&ErrorInfo) == MTK_SUCCESS)
   {
      cout << "\tError Information: " << endl;
      cout << "\t       Error Code: " << ErrorInfo.unErrorCode << endl;
      cout << "\t     Error String: " << ErrorInfo.szErrorString << endl;
      cout << "\t  Additional Info: " << ErrorInfo.szAdditionalInfo << endl;
      cout << endl;
   }
   else
   {
      cout << "mtk_GetErrorInfo() FAILED!!" << endl;
   }
}

char * GetLayoutTypeString(eLB_LAYOUT_TYPE a_eType)
{
   switch (a_eType)
   {
      case eLB_LAYOUT_TYPE_CUSTOM:
         return "CUSTOM";

      case eLB_LAYOUT_TYPE_1_1:
         return "1_1";

      case eLB_LAYOUT_TYPE_4_1:
         return "4_1";
```

```
      default:
         return "UNKNOWN";
   }
}
```

Figure 6. Layout Builder Media Toolkit Example Code Output

```
Layout Builder Media Toolkit Example Code Output
================================================

Create 1_1 Video Layout Successfully... Handle: 256
Created 4_1 Video Layout Successfully... Handle: 512
Created CUSTOM Video Layout Successfully... Handle: 768

**************************************************
** Performing operation on the 1_1 layout type.
**************************************************

lb_AddRegion failed, Expected error due to layout type.

lb_GetRegionList successful... LayoutHandle: 256 RegionCount: 1
   RegionHandle 1: 257
      Display Mode: 2
      Selection Mode: 0
      Priority: 1
      Rect: X:0 Y:0 W:100 H:100

   RegionHandle 1: 257
      lb_SetDisplayMode successfull
      lb_SetSelectionMode successfull
      lb_SetPriority successfull
      lb_SetRect failed, expected error due to layout type

   RegionHandle 1: 257
      Display Mode: 2
      Selection Mode: 1
      Priority: 1
      Rect: X:0 Y:0 W:100 H:100
lb_RemoveRegion failed! Expected error due to layout type


**************************************************
** Performing operation on the 4_1 layout type.
**************************************************

lb_AddRegion failed, Expected error due to layout type.

lb_GetRegionList successful... LayoutHandle: 512 RegionCount: 4
   RegionHandle 1: 513
      Display Mode: 2
      Selection Mode: 1
      Priority: 1
      Rect: X:0 Y:0 W:50 H:50
   RegionHandle 2: 514
      Display Mode: 2
      Selection Mode: 1
      Priority: 1
      Rect: X:0 Y:50 W:50 H:50
   RegionHandle 3: 515
      Display Mode: 2
      Selection Mode: 1
      Priority: 1
      Rect: X:50 Y:0 W:50 H:50
   RegionHandle 4: 516
      Display Mode: 2
```

```
      Selection Mode: 1
      Priority: 1
      Rect: X:50 Y:50 W:50 H:50

   RegionHandle 1: 513
      lb_SetDisplayMode successfull
      lb_SetSelectionMode successfull
      lb_SetPriority successfull
      lb_SetRect failed, expected error due to layout type
   RegionHandle 2: 514
      lb_SetDisplayMode successfull
      lb_SetSelectionMode successfull
      lb_SetPriority successfull
      lb_SetRect failed, expected error due to layout type
   RegionHandle 3: 515
      lb_SetDisplayMode successfull
      lb_SetSelectionMode successfull
      lb_SetPriority successfull
      lb_SetRect failed, expected error due to layout type
   RegionHandle 4: 516
      lb_SetDisplayMode successfull
      lb_SetSelectionMode successfull
      lb_SetPriority successfull
      lb_SetRect failed, expected error due to layout type

   RegionHandle 1: 513
      Display Mode: 2
      Selection Mode: 1
      Priority: 1
      Rect: X:0 Y:0 W:50 H:50
   RegionHandle 2: 514
      Display Mode: 2
      Selection Mode: 1
      Priority: 2
      Rect: X:0 Y:50 W:50 H:50
   RegionHandle 3: 515
      Display Mode: 2
      Selection Mode: 1
      Priority: 3
      Rect: X:50 Y:0 W:50 H:50
   RegionHandle 4: 516
      Display Mode: 2
      Selection Mode: 1
      Priority: 4
      Rect: X:50 Y:50 W:50 H:50
lb_RemoveRegion failed! Expected error due to layout type


**************************************************
** Performing operation on the CUSTOM layout type.
**************************************************
lb_AddRegion successfull

lb_GetRegionList successful... LayoutHandle: 768 RegionCount: 1
   RegionHandle 1: 769
      lb_GetDisplayMode failed
   Error Information:
          Error Code: 0
        Error String: No error
     Additional Info:

      Selection Mode: 0
      Priority: 1
      Rect: X:0 Y:0 W:50 H:50

   RegionHandle 1: 769
      lb_SetDisplayMode successfull
```

```
      lb_SetSelectionMode successfull
      lb_SetPriority successfull
      lb_SetRect successfull

   RegionHandle 1: 769
      Display Mode: 2
      Selection Mode: 1
      Priority: 1
      Rect: X:10 Y:10 W:80 H:80
lb_RemoveRegion successfull

lb_DestroyLayoutTemplate successfull... Handle: 256
lb_DestroyLayoutTemplate successfull... Handle: 512
lb_DestroyLayoutTemplate successfull... Handle: 768
```

6.2 Stream Manipulation Media Toolkit Example Code

Written in the C++ programming language, the example code in Figure 7 illustrates the stream manipulation functionality. The code exercises several functions and data structures that are part of the Dialogic® Media Toolkit API library, including media toolkit, stream manipulation, and overlay builder functions.

It is intended to illustrate how the Dialogic® Media Toolkit API functions and data structures are used in a simple application. It is not intended to be used in a production environment.

The output from the example code is provided in Figure 8 and Figure 9.

Figure 7. Stream Manipulation Media Toolkit Example Code

```
#ifdef WIN32
#else
#include <unistd.h>
#endif

#include "stdio.h"
#include "srllib.h"
#include "mtklib.h"
#include "sm_mtklib.h"
#include "mmlib.h"
#include "ipmlib.h"
#include "port_connect.h"

#define TRUE  1
#define FALSE 0

/*****************************************************************************
 * DEFINE THIS TO CAUSE THE APPLICATION TO USE A BITMAP TO SET THE OVERLAY.
 *****************************************************************************/
//#define BITMAPYUV

/*****************************************************************************
 * DEFINE THIS TO CAUSE OVERLAYS TO BE ADDED TO THE MM DEVICE WITH SEPARATE CALLS
 * TO THE SM LIBRARY.
 *****************************************************************************/
//#define ADD_OVERLAYS_SEPARATELY

/*****************************************************************************
 * DEFINE THIS TO CAUSE THE OVERLAYS TO BE REMOVED FROM THE MM DEVICE WITH
 * SEPARATE CALLS TO THE SM LIBRARY.
 *****************************************************************************/
```

```
//#define REMOVE_OVERLAYS_SEPARATELY


/*************************************************************************
 * THIS DEFINE IS THE NUMBER OF OVERLAYS THAT WILL BE ADDED BY THE
 * BY THE APPLICATION TO THE FIRST MM DEVICE. IF DESIRED, MAKE THIS DEFINE
 * A 3 TO CAUSE A MAXIMUM OVERLAY ERROR TO BE GENERATED DURING
 * THE ADDING OF THE OVERLAYS TO THE MM DEVICE.
 *************************************************************************/
#define OVERLAYCOUNT 2

typedef struct tagTestOverlay
{
    OB_OVERLAY_HANDLE overlayTemplate;
    OB_OVERLAY_HANDLE overlaySnapshot;
    MTK_FRAME_HANDLE  boundingFrameTemplate;
    float x;
    float y;
    eMTK_POSITION_TYPE ePositionType;
    float w;
    float h;
    eMTK_SIZE_TYPE eSizeType;
    MTK_FRAME_HANDLE  boundingFrameSnapshot;
}TestOverlay, *PTestOverlay;

SRL_DEVICE_HANDLE openMMDev(int num);

int connectDevs();
int connectPorts(SRL_DEVICE_HANDLE hDev,
                 PDM_PORT_CONNECT_INFO_LIST pConnectList,
                 PDM_PORT_INFO_LIST pPortInfoList1,
                 PDM_PORT_INFO_LIST pPortInfoList2);
int getPortInfo(SRL_DEVICE_HANDLE hDev,
                 PDM_PORT_INFO_LIST pRPortInfoList,
                 PDM_PORT_INFO_LIST pTPortInfoList);
int createConnectInfoList(PDM_PORT_CONNECT_INFO_LIST pconn_lst,
                          CPDM_PORT_INFO_LIST pport_lst1,
                          CPDM_PORT_INFO_LIST pport_lst2);

const char * getEventString(int evttype);
void outputLastMTKError();
void outputLastDevMgmtError();
int releaseHandles();
const char* translateOverlayResult(eSM_ADD_OVERLAY_RESULT result);
int createMediaItems();
int setOverlayAttrs(TestOverlay* pTestOverlay);
int checkSnapshotOverlayAttrs(TestOverlay* pTestOverlay);
int destroyOverlays();
int handleSMAddOverlay(SRL_DEVICE_HANDLE hDev, long evttype, void *evtdatap);
int handleSMRemoveOverlay(SRL_DEVICE_HANDLE hDev, long evttype, void *evtdatap);
int handleSMRemoveAllOverlays(SRL_DEVICE_HANDLE hDev, long evttype, void * evtdatap);
int waitForDMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType);
int waitForDMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType, int checkData,
                   long* pEventLength, void ** ppEventData, long expectedLength);
int waitForSMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType,
                   long* pReceivedEventType, void ** pevtdatap);
int waitForSMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType,
                   long* pReceivedEventType, void ** pevtdatap,
                   void ** ppuserContext, long expectedLength);
int addAllOverlays();
int addOverlay(int overlayIndex);
int removeOverlay(OB_OVERLAY_HANDLE hOverlay);
int removeOverlaysSeparately();
int removeAllOverlays();
int setOverlaySnapshot(PSM_ADD_OVERLAY_RESULT presult);

DM_PORT_INFO_LIST gmmRPortInfoList;
```

```
DM_PORT_INFO_LIST gmmTPortInfoList;
DM_PORT_INFO_LIST gmm2RPortInfoList;
DM_PORT_INFO_LIST gmm2TPortInfoList;
DM_PORT_CONNECT_INFO_LIST gmmTmm2RConnectList;
DM_PORT_CONNECT_INFO_LIST gmm2TmmRConnectList;

MTK_HANDLE ghYUVImage = MTK_ERROR;
MTK_HANDLE ghJPEGImage = MTK_ERROR;
MTK_HANDLE ghYUVMedia = MTK_ERROR;
MTK_HANDLE ghJPEGMedia = MTK_ERROR;

#ifdef BITMAPYUV
#define BITMAPLENGTH 25344
unsigned char gcBitmapData[BITMAPLENGTH];
#else
const char * g_szYUVFilename = "image.yuv";
#endif

const char * g_szJPEGFilename = "lake_sm.jpg";

TestOverlay gOverlays[] = { {MTK_ERROR, MTK_ERROR, MTK_ERROR,
                             10.0, 15.0, eMTK_POSITION_TYPE_PERCENT,
                             25.0, 20.0, eMTK_SIZE_TYPE_PERCENT,
                             MTK_ERROR},
                            {MTK_ERROR, MTK_ERROR, MTK_ERROR,
                             MTK_JUSTIFY_CENTER , MTK_JUSTIFY_CENTER,
                             eMTK_POSITION_TYPE_JUSTIFICATION,
                             40.0, 40.0, eMTK_SIZE_TYPE_PERCENT,
                             MTK_ERROR},
                            {MTK_ERROR, MTK_ERROR, MTK_ERROR,
                             MTK_JUSTIFY_CENTER , MTK_JUSTIFY_BOTTOM,
                             eMTK_POSITION_TYPE_JUSTIFICATION,
                             40.0, 20.0, eMTK_SIZE_TYPE_PERCENT,
                             MTK_ERROR}
};


unsigned int gOverlayAddRate;
SRL_DEVICE_HANDLE ghMMDev = EMM_ERROR;
SRL_DEVICE_HANDLE ghMM2Dev = EMM_ERROR;

typedef struct
{
    int unEventType;
    char * szEventString;
}EVENT_STRING;

EVENT_STRING gEventStrings[] =
{
    {MMEV_OPEN, "MMEV_OPEN"},
    {MMEV_OPEN_FAIL, "MMEV_OPEN_FAIL"},
    {MMEV_ERROR, "MMEV_ERROR"},
    {SMEV_ADD_OVERLAY, "SMEV_ADD_OVERLAY"},
    {SMEV_ADD_OVERLAY_FAIL, "SMEV_ADD_OVERLAY_FAIL"},
    {SMEV_REMOVE_OVERLAY, "SMEV_REMOVE_OVERLAY"},
    {SMEV_REMOVE_OVERLAY_FAIL, "SMEV_REMOVE_OVERLAY_FAIL"},
    {SMEV_REMOVE_ALL_OVERLAYS, "SMEV_REMOVE_ALL_OVERLAYS"},
    {SMEV_REMOVE_ALL_OVERLAYS_FAIL, "SMEV_REMOVE_ALL_OVERLAYS_FAIL"},
    {SMEV_ERROR, "SMEV_ERROR"},
    {DMEV_PORT_CONNECT, "DMEV_PORT_CONNECT"},
    {DMEV_PORT_CONNECT_FAIL, "DMEV_PORT_CONNECT_FAIL"},
    {DMEV_GET_RX_PORT_INFO, "DMEV_GET_RX_PORT_INFO"},
    {DMEV_GET_TX_PORT_INFO, "DMEV_GET_TX_PORT_INFO"},
    {DMEV_GET_TX_PORT_INFO_FAIL, "DMEV_GET_TX_PORT_INFO_FAIL"},
    {DMEV_GET_RX_PORT_INFO_FAIL, "DMEV_GET_RX_PORT_INFO_FAIL"},
    {DMEV_PORT_DISCONNECT, "DMEV_PORT_DISCONNECT"},
    {DMEV_PORT_DISCONNECT_FAIL, "DMEV_PORT_DISCONNECT_FAIL"}
```

```
};

const char * gszUNKNOWN_EVENT = "unknown event";
#define EVENT_STRING_COUNT  18

int main(int argc, char* argv[])
{
    /*************************************************************************
     * CREATE ALL THE MTK TEMPLATES NEEDED BY THE APPLICATION.
     *************************************************************************/
    if (createMediaItems() == FALSE)
    {
        releaseHandles();
        return -1;
    }

    /*************************************************************************
     * OPEN THE MULTIMEDIA DEVICES TO USE DURING THE APPLICATION
     *************************************************************************/
    if ((ghMMDev = openMMDev(1)) == EMM_ERROR)
    {
        releaseHandles();
        return -1;
    }

    if ((ghMM2Dev = openMMDev(2)) == EMM_ERROR)
    {
        releaseHandles();
        return -1;
    }

    /*************************************************************************
     * ... AND CONNECT THEM.
     *************************************************************************/
    if (!connectDevs())
    {
        releaseHandles();
        return -1;
    }

    /*************************************************************************
     * ADD THE OVERLAYS TO ONE OF THE DEVICES BASED UPON A
     * COMPILATION FLAG. EITHER MAKE A SEPARATE CALL TO THE SM LIBRARAY
     * FOR EACH OVERLAY...
     *************************************************************************/
#ifdef ADD_OVERLAYS_SEPARATELY
    gOverlayAddRate = 1;
    for (int ii = 0; ii < OVERLAYCOUNT; ii++)
    {
        if (addOverlay(ii) == FALSE)
        {
            removeAllOverlays();
            releaseHandles();
            return -1;
        }
    }
#else

    /*************************************************************************
     * ... OR ADD THEM ALL WITH ONE CALL.
     *************************************************************************/
    gOverlayAddRate = OVERLAYCOUNT;
    if(addAllOverlays() == FALSE)
    {
        releaseHandles();
        return -1;
    }
```

```
#endif

    /*************************************************************************
     * HERE WE ARE REMOVING THE OVERLAYS ONE AT A TIME...
     *************************************************************************/
#ifdef REMOVE_OVERLAYS_SEPARATELY
    if(removeOverlaysSeparately() == FALSE)
    {
        releaseHandles();
        return -1;
    }
#else

    /*************************************************************************
     * ... OR ALL AT ONCE, BASED UPON A COMPILATION FLAG.
     *************************************************************************/
    if(removeAllOverlays() == FALSE)
    {
        releaseHandles();
        return -1;
    }
#endif

    /*************************************************************************
     * REMOVE ALL THE MTK TEMPLATES WE CREATED AT THE BEGINNING OF THE
     * APPLICATION.
     *************************************************************************/
    return releaseHandles();
}


int createMediaItems()
{
    /*************************************************************************
     * CREATE A YUV IMAGE TEMPLATE TO REPRESENT THE YUV IMAGE ATTRIBUTES.
     *************************************************************************/
    if ((ghYUVImage = mtk_CreateImageTemplate(eMTK_IMAGE_FORMAT_YUV)) == MTK_ERROR)
    {
        printf("ERROR: returned from createImage\n");
        outputLastMTKError();
        return FALSE;
    }
    else
    {
        printf("Received media YUV image handle: %ld. Setting yuv attributes.\n", ghYUVImage);

        /*********************************************************************
         * SET THE YUV FORMAT TO 4:2:0...
         *********************************************************************/
        eMTK_YUV_IMAGE_FORMAT eFormat = eMTK_YUV_IMAGE_FORMAT_420;
        if (mtk_SetYUVImageFormat(ghYUVImage, eFormat) == MTK_ERROR)
        {
            printf("ERROR: Setting YUV Image format\n");
            outputLastMTKError();
            return FALSE;
        }

        /*********************************************************************
         * ... AND THE SIZE TO 176 BY 144 FOR A QCIF IMAGE.
         *********************************************************************/
        unsigned int w = 176;
        unsigned int h = 144;
        if (mtk_SetYUVImageSize(ghYUVImage, w, h) == MTK_ERROR)
        {
            printf("ERROR: Setting YUV Image size\n");
            outputLastMTKError();
            return FALSE;
```

```
        }

        /*************************************************************************
         * ... TEST THAT WE GET THE VALUES WE JUST SET FOR THE IMAGE.
         *************************************************************************/
        printf("Getting yuv height and width and comparing.\n");
        h = 0;
        w = 0;
        if (mtk_GetYUVImageSize(ghYUVImage, &w, &h) == MTK_ERROR)
        {
            printf("ERROR: getting YUV height and width\n");
            outputLastMTKError();
            return FALSE;
        }
        if ((h != 144)||(w != 176))
        {
            printf("ERROR: YUV height or width values did not match; expected: w(120), h(176)
received: w(%d), h(%d)\n",
                   w, h);
            return FALSE;
        }
    }

#ifdef BITMAPYUV

    /*************************************************************************
     * CREATE A BITMAP TEMPLATE TO CONTAIN THE YUV IMAGE TEMPLATE SNAPSHOT AND
     * THE BITMAP DATA ATTRIBUTES.
     *************************************************************************/
    if ((ghYUVMedia = mtk_CreateBitmapTemplate(ghYUVImage)) == MTK_ERROR)
    {
        printf("ERROR: returned from createMediaFile\n");
        outputLastMTKError();
        return FALSE;
    }
    else
    {
        printf("Received bitmap handle: %ld. Setting memory pointer.\n", ghYUVMedia);

        /*************************************************************************
         * THIS DATA NEEDS TO BE VALID YUV DATA RECEIVED BY THE APPLICATION.
         * HERE WE ARE TESTING THE CREATION OF A BITMAP TEMPLATE.
         *************************************************************************/
        memset((void*)gcBitmapData, 'X', BITMAPLENGTH);
        MTK_BITMAP_DATA bitmapData;
        INIT_MTK_BITMAP_DATA(&bitmapData);
        bitmapData.pucData = gcBitmapData;
        bitmapData.unLength = BITMAPLENGTH;
        if (mtk_SetBitmapData(ghYUVMedia, &bitmapData) == MTK_ERROR)
        {
            printf("ERROR: Setting Bitmap data attribute\n");
            outputLastMTKError();
            return FALSE;
        }
    }

#else

    /*************************************************************************
     * CREATE A MEDIA FILE TEMPLATE TO CONTAIN THE YUV IMAGE SNAPSHOT AND THE
     * FILE ATTRIBUTES.
     *************************************************************************/
    if ((ghYUVMedia = mtk_CreateMediaFileTemplate(ghYUVImage, g_szYUVFilename)) == MTK_ERROR)
    {
        printf("ERROR: returned from create media file for yuv\n");
        outputLastMTKError();
        return FALSE;
```

```
    }
    else
    {
        printf("Received yuv media file handle: %d\n", ghYUVMedia);
    }
#endif

    /*************************************************************************
     * CREATE A JPEG IMAGE TEMPLATE TO REPRESENT THE JPEG IMAGE ATTRIBUTES.
     ************************************************************************/
    if ((ghJPEGImage = mtk_CreateImageTemplate(eMTK_IMAGE_FORMAT_JPEG)) == MTK_ERROR)
    {
        printf("ERROR: returned from createImage\n");
        outputLastMTKError();
        return FALSE;
    }
    else
    {
        printf("Received media JPEG image handle: %ld.\n", ghJPEGImage);
    }

    /*************************************************************************
     * CREATE A MEDIA FILE TEMPLATE TO CONTAIN THE JPEG IMAGE SNAPSHOT AND THE
     * FILE ATTRIBUTES.
     ************************************************************************/
    if ((ghJPEGMedia = mtk_CreateMediaFileTemplate(ghJPEGImage, g_szJPEGFilename)) == MTK_ERROR)
    {
        printf("ERROR: returned from create media file for jpeg\n");
        outputLastMTKError();
        return FALSE;
    }
    else
    {
        printf("Received jpeg media file handle: %d\n", ghJPEGMedia);
    }

    /*************************************************************************
     * CREATE THE OVERLAY TEMPLATES.
     ************************************************************************/
    for (int ii = 0; ii < OVERLAYCOUNT; ii++)
    {
        /*********************************************************************
         * CREATE A BOUNDING FRAME FOR EACH OVERLAY TEMPLATE...
         ********************************************************************/
        printf("Creating frame template\n");
        if ((gOverlays[ii].boundingFrameTemplate = mtk_CreateFrameTemplate())
                                                                == MTK_ERROR)
        {
            printf("ERROR: Creating frame template\n");
            outputLastMTKError();
            return FALSE;
        }

        /*********************************************************************
         * ... SET THE FRAME POSITION...
         ********************************************************************/
        printf("Setting frame [%ld] position:\n\tx: [%f]; y: [%f]; pos enum[%d]\n",
               gOverlays[ii].boundingFrameTemplate,
               gOverlays[ii].x,
               gOverlays[ii].y,
               gOverlays[ii].ePositionType);
        if (mtk_SetFramePosition(gOverlays[ii].boundingFrameTemplate,
                                 gOverlays[ii].x,
                                 gOverlays[ii].y,
                                 gOverlays[ii].ePositionType) == MTK_ERROR)
        {
            printf("ERROR: Setting frame template position values\n");
```

```
            outputLastMTKError();
            return FALSE;
        }

        /*************************************************************************
         * ... SET THE FRAME SIZE.
         *************************************************************************/
        printf("Setting frame [%ld] size:\n\tw: [%f]; h: [%f]; size enum[%d]\n",
               gOverlays[ii].boundingFrameTemplate,
               gOverlays[ii].w,
               gOverlays[ii].h,
               gOverlays[ii].eSizeType);
        // set the frame size
        if (mtk_SetFrameSize(gOverlays[ii].boundingFrameTemplate,
                             gOverlays[ii].w,
                             gOverlays[ii].h,
                             gOverlays[ii].eSizeType) == MTK_ERROR)
        {
            printf("ERROR: Setting frame template size values\n");
            outputLastMTKError();
            return FALSE;
        }

        MTK_HANDLE hMedia;

        if (ii%2)
        {
            hMedia = ghJPEGMedia;
        }
        else
        {
            hMedia = ghYUVMedia;
        }

        /*************************************************************************
         * CREATE THE OVERLAY TEMPLATE ALTERNATING BETWEEN THE JPEG AND YUV
         * MEDIA...
         *************************************************************************/
        if ((gOverlays[ii].overlayTemplate =
                                ob_CreateImageOverlayTemplate(hMedia)) == MTK_ERROR)
        {
            printf("ERROR: returned from createImageOverlay for %d\n", ii);
            outputLastMTKError();
            return FALSE;
        }
        else
        {
            printf("Received overlay handle [%ld] for overlay index [%d]\n",
                   gOverlays[ii].overlayTemplate, ii);
            /*****************************************************************
             * ... AND SET THE OVERLAY ATTRIBUTES.
             *****************************************************************/
            if (setOverlayAttrs(&gOverlays[ii]) == FALSE)
            {
                return FALSE;
            }
        }
    }

    return TRUE;
}


int setOverlayAttrs(TestOverlay* pTestOverlay)
{
    unsigned int unDuration = MTK_INFINITE;
```

```
    /*************************************************************************
     * ... SET THE DURATION TO BE INFINITE...
     ************************************************************************/
    printf("Setting overlay duration\n");
    if (ob_SetOverlayDuration(pTestOverlay->overlayTemplate, unDuration) == MTK_ERROR)
    {
        printf("ERROR: Setting overlay duration\n");
        outputLastMTKError();
        return FALSE;
    }

    /*************************************************************************
     * ... THE BOUNDING FRAME...
     ************************************************************************/
    printf("Setting bounding frame for the overlay template\n");
    // set the template frame getting the handle of the snapshot frame
    if ((pTestOverlay->boundingFrameSnapshot =
               ob_SetOverlayBoundingFrame(pTestOverlay->overlayTemplate,
                                          pTestOverlay->boundingFrameTemplate))
                                                                == MTK_ERROR)
    {
        printf("ERROR: Setting overlay template frame\n");
        outputLastMTKError();
        return FALSE;
    }

    /*************************************************************************
     * ... AND THE FILL STYLE. WE ARE USING RESIZE TO FIT SINCE THAT IS THE
     * ONLY STYLE CURRENTLY SUPPORTED.
     ************************************************************************/
    printf("Setting overlay fill style\n");
    if (ob_SetOverlayFillStyle(pTestOverlay->overlayTemplate,
                                          eOB_FILL_STYLE_RESIZE_TO_FIT) == MTK_ERROR)
    {
        printf("ERROR: Setting overlay fill style\n");
        outputLastMTKError();
        return FALSE;
    }

    return TRUE;
}


/*****************************************************************************
 * TEST THAT THE SNAPSHOT ATTRIBUTES ARE THE SAME ONES THAT WE SET IN THE
 * ORIGINAL TEMPLATES.
 ****************************************************************************/
int checkSnapshotOverlayAttrs(TestOverlay* pTestOverlay)
{
    /*************************************************************************
     * GET THE BOUNDING FRAME HANDLE FOR THE OVERLAY SNAPSHOT. THIS WILL BE A
     * UNIQUE BOUNDING FRAME HANDLE SINCE IT IS ALSO A SNAPSHOT OF THE FRAME
     * REFERENCED BY THE OVERLAY TEMPLATE AT THE TIME OF THE ADD.
     ************************************************************************/
    MTK_FRAME_HANDLE hSnapshotOverlayFrame;
    if (ob_GetOverlayBoundingFrame(pTestOverlay->overlaySnapshot,
                                   &hSnapshotOverlayFrame) == MTK_ERROR)
    {
        printf("ERROR: getting snapshot overlay frame handle\n");
        outputLastMTKError();
        return FALSE;
    }

    /*************************************************************************
     * GET THE POSITION AND SIZE OF THE SNAPSHOT FRAME AND COMPARE IT TO THOSE
     * WE ORIGINALLY SET ON THE FRAME TEMPLATE.
     ************************************************************************/
```

```
    printf("Checking bounding frame position and size for the overlay snapshot against set
values\n");
    float x;
    float y;
    eMTK_POSITION_TYPE ePositionType;
    if (mtk_GetFramePosition(hSnapshotOverlayFrame,
                             &x,
                             &y,
                             &ePositionType) == MTK_ERROR)
    {
        printf("ERROR: getting frame position values\n");
        outputLastMTKError();
        return FALSE;
    }

    if ((pTestOverlay->x == x) &&
        (pTestOverlay->y == y) &&
        (pTestOverlay->ePositionType == ePositionType))
    {
        printf("Snapshot overlay position information matches settings.\n");
    }
    else
    {
        printf("ERROR: Snapshot overlay position information does not match settings.\n");
        return FALSE;
    }

    float h;
    float w;
    eMTK_SIZE_TYPE eSizeType;
    if (mtk_GetFrameSize(hSnapshotOverlayFrame,
                         &w,
                         &h,
                         &eSizeType) == MTK_ERROR)
    {
        printf("ERROR: Setting frame template size values\n");
        outputLastMTKError();
        return FALSE;
    }

    if ((pTestOverlay->w == w) &&
        (pTestOverlay->h == h) &&
        (pTestOverlay->eSizeType == eSizeType))
    {
        printf("Snapshot overlay size information matches settings.\n");
    }
    else
    {
        printf("ERROR: Snapshot overlay size information does not match settings.\n");
        return FALSE;
    }

    /*************************************************************************
     * NOW GET THE OVERLAY FILL STYLE OF THE SNAPSHOT TEMPLATE AND COMPARE IT
     * TO THE VALUE SET ON THE ORIGINAL OVERLAY TEMPLATE.
     *************************************************************************/
    eOB_FILL_STYLE eFillStyle;
    printf("Checking overlay fill style\n");
    if (ob_GetOverlayFillStyle(pTestOverlay->overlaySnapshot, &eFillStyle) == MTK_ERROR)
    {
        printf("ERROR: getting overlay fill style\n");
        outputLastMTKError();
        return FALSE;
    }

    if (eFillStyle == eOB_FILL_STYLE_RESIZE_TO_FIT)
    {
```

```
        printf("Snapshot overlay fill style matches set value.\n");
    }
    else
    {
        printf("ERROR: Snapshot overlay fill style does not match set value.\n");
        return FALSE;
    }

    return TRUE;
}


const char * getEventString(int evttype)
{
    const char* pReturnString = gszUNKNOWN_EVENT;

    for (int ii = 0; ii < EVENT_STRING_COUNT; ii++)
    {
        if (gEventStrings[ii].unEventType == evttype)
        {
            pReturnString = gEventStrings[ii].szEventString;
            break;
        }
    }
    return pReturnString;
}


/*****************************************************************************
 * FOR MTK LAST ERROR RETRIEVAL UPON FUNCTION FAILURES.
 *****************************************************************************/
void outputLastMTKError()
{
    MTK_ERROR_INFO errorInfo;
    INIT_MTK_ERROR_INFO(&errorInfo);

    if(mtk_GetErrorInfo(&errorInfo) == MTK_ERROR)
    {
        printf("ERROR getting last mtk error\n");
    }
    else
    {
        printf("MTK Last Error: unErrorCode[%d], szErrorString[%s], szAdditionalInfo[%s]\n",
               errorInfo.unErrorCode, errorInfo.szErrorString, errorInfo.szAdditionalInfo);
    }
}


/*****************************************************************************
 * RETRIEVE THE SM EVENT RELATED DATA
 *****************************************************************************/
int waitForSMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType,
                   long* pReceivedEventType, void ** pevtdatap,
                   void ** puserContext, long expectedLength)
{
    if (sr_waitevt(10000) == -1)
    {
        printf("sr_waitevt() failure\n");
        return FALSE;
    }

    *pReceivedEventType = sr_getevttype();
    long evtdev         = sr_getevtdev();
    long evtlen         = sr_getevtlen();
    *pevtdatap          = sr_getevtdatap();

    if (expectedLength != 0)
```

```
    {
        if (expectedLength  == evtlen)
        {
            printf("Event length matches expected\n");
        }
        else
        {
            printf("ERROR: Event length(%ld) does not match expected(%ld)\n",
                   evtlen, expectedLength);
        }
    }

    if (puserContext)
    {
        *puserContext   = sr_getUserContext();
    }

    if (evtdev != hDev)
    {
        printf("event [0x%x] for unknown device handle [%d]\n", evtdev);
        return FALSE;
    }

    if (expectedEventType == *pReceivedEventType)
    {
        printf("Received expected event [%s]\n", getEventString(expectedEventType));
        return TRUE;
    }
    else
    {
        printf("ERROR: Received unexpected event [%s] while waiting for %s\n",
               getEventString(*pReceivedEventType),
               getEventString(expectedEventType));
        return FALSE;
    }

    return TRUE;
}


/*****************************************************************************
 * RETRIEVE THE SM EVENT RELATED DATA
 *****************************************************************************/
int waitForSMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType,
                   long* pReceivedEventType, void ** pevtdatap)
{
    return waitForSMEvent(hDev, expectedEventType, pReceivedEventType,
                          pevtdatap, NULL, 0);
}


/*****************************************************************************
 * RETRIEVE THE DEVICE MANAGEMENT (DML) EVENT RELATED DATA
 *****************************************************************************/
int waitForDMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType)
{
    long evtlen;
    void * pevtdata;

    return waitForDMEvent(hDev, expectedEventType, FALSE, &evtlen, &pevtdata, 0);
}

/*****************************************************************************
 * RETRIEVE THE DEVICE MANAGEMENT (DML) EVENT RELATED DATA
 *****************************************************************************/
int waitForDMEvent(SRL_DEVICE_HANDLE hDev, long expectedEventType, int checkData,
                   long* pEventLength, void ** ppEventData, long expectedLength)
```

```
{
    if (sr_waitevt(10000) == -1)
    {
        printf("wait event FALSE\n");
        return FALSE;
    }

    long evttype = sr_getevttype();
    long evtdev = sr_getevtdev();
    *pEventLength = sr_getevtlen();
    *ppEventData = sr_getevtdatap();

    if (evtdev != hDev)
    {
        printf("event [%s] for unknown device handle [%d]\n", getEventString(evttype), evtdev);
        return FALSE;
    }

    if ((checkData) && (*ppEventData == NULL))
    {
        printf("ERROR: Dev Mgmt Event data pointer is NULL\n");
        return FALSE;
    }

    if (expectedEventType == evttype)
    {
        printf("Received expected DM event: %s\n", getEventString(expectedEventType));
    }
    else
    {
        printf("ERROR: Received %s while waiting for %s\n", getEventString(evttype),
getEventString(expectedEventType));
        return FALSE;
    }

    if (expectedLength != 0)
    {
        if (expectedLength  == *pEventLength)
        {
            printf("Event length matches expected\n");
        }
        else
        {
            printf("ERROR: Event length(%ld) does not match expected(%ld)\n",
                   *pEventLength, expectedLength);
        }
    }

    return TRUE;
}


/*************************************************************************
 * FOR DML LAST ERROR RETRIEVAL UPON FUNCTION FAILURES.
 *************************************************************************/
void outputLastDevMgmtError()
{
    DEV_ERRINFO errorInfo;

    if(dev_ErrorInfo(&errorInfo) == -1)
    {
        printf("ERROR getting last dev mgmt error\n");
    }
    else
    {
        printf("Dev Mgmt Last Error: dev_ErrValue[%d], dev_SubSystemErrValue[%d],
dev_Msg[%s]\n",
```

```
                errorInfo.dev_ErrValue, errorInfo.dev_SubSystemErrValue, errorInfo.dev_Msg);
    }
}


/*****************************************************************************
 * RELEASE ALL MTK TEMPLATE HANDLES AND CLOSE THE MM DEVICES.
 *****************************************************************************/
int releaseHandles()
{
    int ret = 0;

    if (ghYUVMedia != MTK_ERROR)
    {
        /*************************************************************************
         * DESTROY THE YUV MEDIA CONTAINER TEMPLATE
         *************************************************************************/
        if (mtk_DestroyMediaTemplate(ghYUVMedia) == MTK_ERROR)
        {
            printf("ERROR returned from mtk_DestroyMediaItem() for YUV media template\n");
            outputLastMTKError();
            ret = -1;
        }
        else
        {
            printf("Successfully destroyed YUV media template\n");
        }
    }

    if (ghJPEGMedia != MTK_ERROR)
    {
        /*************************************************************************
         * DESTROY THE JPEG MEDIA CONTAINER TEMPLATE
         *************************************************************************/
        if (mtk_DestroyMediaTemplate(ghJPEGMedia) == MTK_ERROR)
        {
            printf("ERROR returned from mtk_DestroyMediaItem() for jpeg media template\n");
            outputLastMTKError();
            ret = -1;
        }
        else
        {
            printf("Successfully destroyed media template\n");
        }
    }

    if (ghYUVImage != MTK_ERROR)
    {
        /*************************************************************************
         * DESTROY THE YUV IMAGE TEMPLATE
         *************************************************************************/
        if (mtk_DestroyMediaTemplate(ghYUVImage) == MTK_ERROR)
        {
            printf("ERROR returned from mtk_DestroyMediaItem() for yuv image\n");
            outputLastMTKError();
            ret = -1;
        }
        else
        {
            printf("Successfully destroyed yuv image\n");
        }
    }

    if (ghJPEGImage != MTK_ERROR)
    {
        /*************************************************************************
         * DESTROY THE JPEG IMAGE TEMPLATE
```

```
     ***************************************************************************/
    if (mtk_DestroyMediaTemplate(ghJPEGImage) == MTK_ERROR)
    {
        printf("ERROR returned from mtk_DestroyMediaItem() for jpeg image\n");
        outputLastMTKError();
        ret = -1;
    }
    else
    {
        printf("Successfully destroyed jpeg image\n");
    }
}

/***************************************************************************
 * DESTROY EACH FRAME TEMPLATE...
 ***************************************************************************/
for (int ii = 0; ii < OVERLAYCOUNT; ii++)
{
    if (gOverlays[ii].boundingFrameTemplate != MTK_ERROR)
    {
        if ((mtk_DestroyFrameTemplate(gOverlays[ii].boundingFrameTemplate))
                                                                    == MTK_ERROR)
        {
            printf("ERROR returned from  creating frame template\n");
            outputLastMTKError();
            ret = -1;
        }
        else
        {
            printf("Successfully destroyed frame template[%ld]\n",
                   gOverlays[ii].boundingFrameTemplate);
        }
    }
}

/***************************************************************************
 * ... AND THE OVERLAY TEMPLATES
 ***************************************************************************/
ret = destroyOverlays();

/***************************************************************************
 * CLOSE THE MM DEVICES
 ***************************************************************************/
if (ghMMDev != EMM_ERROR)
{
    if (mm_Close(ghMMDev, NULL) == EMM_ERROR)
    {
        printf("ERROR returned from mm_Close(%ld)\n", ghMMDev);
        ret = -1;
    }
    else
    {
        printf("Successfully closed mm device\n");
    }
}

if (ghMM2Dev != EMM_ERROR)
{
    if (mm_Close(ghMM2Dev, NULL) == EMM_ERROR)
    {
        printf("ERROR returned from mm_Close(%ld)\n", ghMM2Dev);
        ret = -1;
    }
    else
    {
        printf("Successfully closed mm2 device\n");
    }
```

```
    }

    return ret;
}


/*****************************************************************************
 * DESTROY THE OVERLAY TEMPLATES
 *****************************************************************************/
int destroyOverlays()
{
    int ret = 0;
    for (int ii = 0; ii < OVERLAYCOUNT; ii++)
    {
        /*************************************************************************
         * IF THE TEMPLATE WAS CREATED...
         *************************************************************************/
        if (gOverlays[ii].overlayTemplate != MTK_ERROR)
        {
            /*************************************************************************
             * ... DESTROY IT.
             *************************************************************************/
            if (ob_DestroyOverlayTemplate(gOverlays[ii].overlayTemplate) == MTK_ERROR)
            {
                printf("ERROR returned from ob_DestroyOverlay() for overlay[%ld]\n",
                       gOverlays[ii].overlayTemplate);
                outputLastMTKError();
                ret = -1;
            }
            else
            {
                printf("Successfully destroyed overlay [%ld]\n", gOverlays[ii].overlayTemplate);
            }
        }
    }
    return ret;
}


/*****************************************************************************
 * OPEN AN MM DEVICE.
 *****************************************************************************/
SRL_DEVICE_HANDLE openMMDev(int num)
{
    printf("Opening mm device %d\n", num);
    char buf[20];

    sprintf(buf, "mmB1C%d", num);
    SRL_DEVICE_HANDLE hMMDev = mm_Open(buf, NULL, NULL);

    if (hMMDev == EMM_ERROR)
    {
        printf("ERROR: mm_Open() failure\n");
        return EMM_ERROR;
    }

    if (sr_waitevt(10000) == -1)
    {
        printf("ERROR: sr_waitevt failure\n");
        return EMM_ERROR;
    }

    MM_METAEVENT eventInfo;
    mm_GetMetaEvent(&eventInfo);
    if (eventInfo.evtdev != hMMDev)
    {
        printf("ERROR: event for unknown device handle [%ld]\n", eventInfo.evtdev);
```

```
        return EMM_ERROR;
    }
    else
    {
        if (eventInfo.evttype == MMEV_OPEN)
        {
            printf("MMEV_OPEN event; received handle [%ld]\n", eventInfo.evtdev);
        }
        else
        {
            printf("ERROR: %s event for handle [%d]\n", eventInfo.evttype, eventInfo.evtdev);
            return EMM_ERROR;
        }
    }
    return hMMDev;
}


/*********************************************************************
 * CREATES THE CONNECTION INFO LIST FOR DEV_PORT_CONNECT.
 *********************************************************************/
int createConnectInfoList(PDM_PORT_CONNECT_INFO_LIST pconn_lst,
                          CPDM_PORT_INFO_LIST ptrans_port_lst,
                          CPDM_PORT_INFO_LIST precv_port_lst)
{
     int bFound = FALSE;
     unsigned int jj, ii, nmatches = 0;
     DM_PORT_MEDIA_TYPE type_rx, type_tx;

     INIT_DM_PORT_CONNECT_INFO_LIST(pconn_lst);
    printf("Transmit port list count: %d\n", ptrans_port_lst->unCount);
     // Loop through all transmit ports of 1st device
     for (ii = 0; ii < ptrans_port_lst->unCount; ii++)
    {
          type_tx = ptrans_port_lst->port_info[ii].port_media_type;
        printf("Transmission port list type_tx: %d\n", type_tx);
        bFound = FALSE;

          // find appropriate RX port on 2nd device
        printf("Receive port list count: %d\n", precv_port_lst->unCount);
          for (jj = 0; jj < precv_port_lst->unCount; jj++)
        {
               type_rx = precv_port_lst->port_info[jj].port_media_type;
            printf("Receive port list type_rx: %ld\n", type_rx);
               if (type_tx == type_rx)
            {
                   bFound = TRUE;
                   break;
              }
         }

         if (!bFound) {
             continue;
         }

         // create element of connect list
         DM_PORT_CONNECT_INFO& info = pconn_lst->port_connect_info[nmatches];
        INIT_DM_PORT_CONNECT_INFO(&info);
        info.unFlags = DMFL_TRANSCODE_ON;
         info.port_info_tx = ptrans_port_lst->port_info[ii];
         info.port_info_rx = precv_port_lst->port_info[jj];
         nmatches++;
     }

     pconn_lst->unCount = nmatches;
     return nmatches;
}
```

```
/*****************************************************************
 * RETRIEVES PORT INFO FOR THE MM DEVICE.
 *****************************************************************/
int getPortInfo(SRL_DEVICE_HANDLE hDev, PDM_PORT_INFO_LIST pRPortInfoList, PDM_PORT_INFO_LIST
pTPortInfoList)
{
    printf("Calling dev_GetTransmitPortInfo() for device handle [%d]\n", hDev);
    if ((dev_GetTransmitPortInfo(hDev, NULL)) == -1)
    {
        printf("dev_GetTransmitPortInfo error\n");
        outputLastDevMgmtError();
        return FALSE;
     }

    long evtlen;
    void * pevtdata;

    if (waitForDMEvent(hDev, DMEV_GET_TX_PORT_INFO, TRUE, &evtlen, &pevtdata,
sizeof(DM_PORT_INFO_LIST)))
    {
        memcpy(pTPortInfoList, pevtdata, sizeof(DM_PORT_INFO_LIST));
    }
    else
    {
        printf("dev_GetTransmitPortInfo() failure for handle [%d]\n", hDev);
        return FALSE;
    }

    printf("Calling dev_GetReceivePortInfo() for device handle [%d]\n", hDev);
     if ((dev_GetReceivePortInfo(hDev, NULL) == -1))
    {
        printf("dev_GetReceivePortInfo error\n", hDev);
        outputLastDevMgmtError();
        return FALSE;
     }

    if (waitForDMEvent(hDev, DMEV_GET_RX_PORT_INFO, TRUE, &evtlen, &pevtdata,
sizeof(DM_PORT_INFO_LIST)))
    {
        memcpy(pRPortInfoList, pevtdata, sizeof(DM_PORT_INFO_LIST));
    }
    else
    {
        printf("dev_GetReceivePortInfo() failure for handle [%d]\n", hDev);
        return FALSE;
    }

    return TRUE;
}


/*****************************************************************
 * GETS THE PORT INFO FOR EACH DEVICE AND CONNECTS THEM.
 *****************************************************************/
int connectDevs()
{
    printf("Calling getPortInfo() for mm device...\n");
    if(!getPortInfo(ghMMDev, &gmmRPortInfoList, &gmmTPortInfoList))
    {
        printf("failing due to getPortInfo() FALSE\n");
        return FALSE;
    }

    printf("Calling getPortInfo() for mm2 device...\n");
    if(!getPortInfo(ghMM2Dev, &gmm2RPortInfoList, &gmm2TPortInfoList))
```

```
    {
        printf("failing due to getPortInfo() FALSE\n");
        return FALSE;
    }

    printf("Calling connectPorts() for mm tx and mm2 rx ports...\n");
    if(!connectPorts(ghMMDev, &gmmTmm2RConnectList,
                    &gmmTPortInfoList, &gmm2RPortInfoList))
    {
        printf("failing due to connectPorts() failure\n");
        return FALSE;
    }

    printf("Calling connectPorts() for mm2 tx and mm rx ports...\n");
    if(!connectPorts(ghMM2Dev, &gmm2TmmRConnectList,
                    &gmm2TPortInfoList, &gmmRPortInfoList))
    {
        printf("failing due to connectPorts() failure\n");
        return FALSE;
    }
    return TRUE;
}


/*****************************************************************************
 * CREATES THE CONNECTION INFO LIST FROM THE PORT INFO FOR THE MM DEVICE AND USES
 * THAT TO CONNECT THEM.
 *****************************************************************************/
int connectPorts(SRL_DEVICE_HANDLE hDev,
                PDM_PORT_CONNECT_INFO_LIST pConnectList,
                PDM_PORT_INFO_LIST pTransPortInfoList,
                PDM_PORT_INFO_LIST pRecvPortInfoList)
{
    printf("Calling createConnectInfoList() for dev handle [%ld]...\n", hDev);
    if(createConnectInfoList(pConnectList,
                            pTransPortInfoList,
                            pRecvPortInfoList) == 0)
    {
        printf("connectPorts() failing due to createConnectInfoList() FALSE\n");
        return FALSE;
    }

    printf("Calling dev_PortConnect() for dev handle [%ld]...\n", hDev);
if (dev_PortConnect(hDev, pConnectList, NULL) == -1)
    {
        printf("dev_PortConnect error\n");
        outputLastDevMgmtError();
        return FALSE;
}

    return waitForDMEvent(hDev, DMEV_PORT_CONNECT);
}


/*****************************************************************************
 * ADDS ALL THE OVERLAYS TO THE FIRST MM DEVICE WITH ONE CALL TO THE SM LIBRARY.
 *****************************************************************************/
int addAllOverlays()
{
    SM_ADD_OVERLAY_LIST overlayList;
    INIT_SM_ADD_OVERLAY_LIST(&overlayList);
    overlayList.unCount = OVERLAYCOUNT;

    /*************************************************************************
     * LOOP THROUGH THE OVERLAY LIST AND SET THE TEMPLATE HANDLES AND
     * THE DIRECTION...
     *************************************************************************/
```

```
    for (int ii = 0; ii < OVERLAYCOUNT; ii++)
    {
        gOverlays[ii].overlaySnapshot = MTK_ERROR;
        overlayList.addOverlays[ii].eDirection = eSM_OVERLAY_DIRECTION_DEVICE;
        overlayList.addOverlays[ii].hOverlay = gOverlays[ii].overlayTemplate;
    }

    /*************************************************************************
     * ... AND CALL THE SM LIBRARY TO PERFORM THE ADD OF ALL THE OVERLAYS.
     *************************************************************************/
    printf("Calling sm_AddOverlay for %d overlays\n", OVERLAYCOUNT);
    if(sm_AddOverlays(ghMMDev, &overlayList, &ghMMDev) == MTK_ERROR)
    {
        printf("ERROR returned from sm_AddOverlay\n");
        outputLastMTKError();
        return FALSE;
    }

    printf("Waiting for add overlay completion event...\n");

    long evttype;
    void * evtdatap = NULL;
    void * puserContext = NULL;

    /*************************************************************************
     * WAIT FOR THE SMEV_ADD_OVERLAY EVENT.
     *************************************************************************/
    if (waitForSMEvent(ghMMDev,
                       SMEV_ADD_OVERLAY,
                       &evttype,
                       &evtdatap,
                       &puserContext,
                       sizeof(SM_ADD_OVERLAY_RESULT_LIST)))
    {
        if (puserContext == &ghMMDev)
        {
            printf("successful event user context match\n");
            return handleSMAddOverlay(ghMMDev, evttype, evtdatap);
        }
        else
        {
            printf("failure: event user context does not match input\n");
            handleSMAddOverlay(ghMMDev, evttype, evtdatap);
            return FALSE;
        }
    }
    else
    {
        printf("sm_AddOverlays() failure [%d]\n", ghMMDev);
        handleSMAddOverlay(ghMMDev, evttype, evtdatap);
        return FALSE;
    }
}

/*************************************************************************
 * ADD EACH OVERLAY TO MM DEVICE ONE WITH A SEPARATE CALL TO THE SM LIBRARY.
 *************************************************************************/
int addOverlay(int overlayIndex)
{
    SM_ADD_OVERLAY_LIST overlayList;
    INIT_SM_ADD_OVERLAY_LIST(&overlayList);
    overlayList.unCount = 1;

    /*************************************************************************
     * CREATE AN OVERLAY LIST AND SET THE TEMPLATE HANDLE AND
     * THE DIRECTION FOR THE ONE OVRELAY...
     *************************************************************************/
```

```
    gOverlays[overlayIndex].overlaySnapshot = MTK_ERROR;
    overlayList.addOverlays[0].eDirection = eSM_OVERLAY_DIRECTION_DEVICE;
    overlayList.addOverlays[0].hOverlay = gOverlays[overlayIndex].overlayTemplate;

    /*************************************************************************
     * ... AND CALL THE SM LIBRARY TO PERFORM THE ADD OF THE SINGLE OVERLAY.
     *************************************************************************/
    printf("Calling sm_AddOverlay for overlay handle[%ld]\n",
           gOverlays[overlayIndex].overlayTemplate);

    if(sm_AddOverlays(ghMMDev, &overlayList, &ghMMDev) == MTK_ERROR)
    {
        printf("ERROR returned from sm_AddOverlay\n");
        outputLastMTKError();
        return FALSE;
    }

    printf("Waiting for add overlay completion event...\n");
    long evttype = 0;
    void * evtdatap = NULL;
    void * puserContext = NULL;

    /*************************************************************************
     * WAIT FOR THE SMEV_ADD_OVERLAY EVENT.
     *************************************************************************/
    if (waitForSMEvent(ghMMDev,
                       SMEV_ADD_OVERLAY,
                       &evttype,
                       &evtdatap,
                       &puserContext,
                       sizeof(SM_ADD_OVERLAY_RESULT_LIST)))
    {
        if (puserContext == &ghMMDev)
        {
            printf("successful event user context match\n");
            return handleSMAddOverlay(ghMMDev, evttype, evtdatap);
        }
        else
        {
            printf("failure: event user context does not match input\n");
            handleSMAddOverlay(ghMMDev, evttype, evtdatap);
            return FALSE;
        }
    }
    else
    {
        printf("sm_AddOverlays() failure [%d]\n", ghMMDev);
        handleSMAddOverlay(ghMMDev, evttype, evtdatap);
        return FALSE;
    }
}

/*************************************************************************
 * PROCESS SM LIBRARY ADD OVERLAY RELATED EVENTS.
 *************************************************************************/
int handleSMAddOverlay(SRL_DEVICE_HANDLE hDev, long evttype, void *evtdatap)
{
    int ret = FALSE;

    switch(evttype)
    {
    case SMEV_ADD_OVERLAY:
        /*********************************************************************
         * GET THE OVERLAY SNAPSHOT HANDLE AND SAVE IT FOR THE LATER CALL
         * TO REMOVE.
         *********************************************************************/
        if (evtdatap)
```

```
        {
            /*************************************************************
             * MAKE SURE WE GET THE SAME OVERLAY COUNT WE SENT DOWN.
             *************************************************************/
            PSM_ADD_OVERLAY_RESULT_LIST paddOverlay =
                                        (PSM_ADD_OVERLAY_RESULT_LIST)evtdatap;
            if (paddOverlay->unCount != gOverlayAddRate)
            {
                printf("ERROR: add overlay count not equal to %d; value[%d]\n",
                       gOverlayAddRate, paddOverlay->unCount);

            }
            else
            {
                /*********************************************************
                 * LOOP THROUGH THE OVERLAY ADD RESULT LIST AND SAVE THE
                 * OVERLAY SNAPSHOT HANDLE FOR THE LATER CALL TO REMOVE OVERLAY.
                 *********************************************************/
                ret = TRUE;
                PSM_ADD_OVERLAY_RESULT presult = paddOverlay->overlayResults;
                for (unsigned int ii = 0; ii < gOverlayAddRate; ii++, presult++)
                {
                    if (setOverlaySnapshot(presult) == FALSE)
                    {
                        ret = FALSE;
                    }
                }
            }
        }
        else
        {
            printf("ERROR: NULL event data for SMEV_ADD_OVERLAY event\n");
        }
        break;

    case SMEV_ADD_OVERLAY_FAIL:
        /*************************************************************
         * LOG THE ERROR AND THE FAILING OVERLAY TEMPLATE HANDLES.
         *************************************************************/
        if (evtdatap)
        {
            PSM_ADD_OVERLAY_RESULT_LIST paddOverlay = (PSM_ADD_OVERLAY_RESULT_LIST)evtdatap;
            PSM_ADD_OVERLAY_RESULT presult = paddOverlay->overlayResults;
            for (unsigned int ii = 0; ii < paddOverlay->unCount; ii++, presult++)
            {
                printf("failing overlay template handle[%d]; overlayResult[%s]\n",
                       presult->hOverlayTemplate,
                       translateOverlayResult(presult->result));
            }
        }
        else
        {
            printf("ERROR: NULL event data for SMEV_ADD_OVERLAY_FAIL event\n");
        }
        break;

    default:
        printf("ERROR: non-SM event received\n");
    };

    return ret;
}


/*************************************************************************
 * SAVES THE OVERLAY SNAPSHOT HANDLES FOR LATER USE WITH SM REMOVE OVERLAY.
 *************************************************************************/
```

```
int setOverlaySnapshot(PSM_ADD_OVERLAY_RESULT presult)
{
    int ret = FALSE;

    /*************************************************************************
     * LOOP THROUGH THE OVERLAYS TEMPLATES LOOKING FOR THE TEMPLATE HANDLE IN
     * THE RESULT.
     *************************************************************************/
    for (int ii = 0; ii < OVERLAYCOUNT; ii++)
    {
        if(presult->hOverlayTemplate == gOverlays[ii].overlayTemplate)
        {
            /*********************************************************************
             * ONCE WE HAVE FOUND A MATCH, MAKE SURE WE HAVE NOT ALREADY SAVED
             * THIS TEMPLATE SNAPSHOT...
             *********************************************************************/
            if (gOverlays[ii].overlaySnapshot == MTK_ERROR)
            {
                gOverlays[ii].overlaySnapshot = presult->hOverlaySnapshot;
                printf("Received overlay: template handle [%ld]; snapshot handle [%ld];
result[%s]\n",
                       presult->hOverlayTemplate, presult->hOverlaySnapshot,
translateOverlayResult(presult->result));

                /*************************************************************
                 * ... AND, AS A TEST, CHECK THAT THE ATTRIBUTES OF THE
                 * SNAPSHOT MATCH THOSE WE SET.
                 *************************************************************/
                checkSnapshotOverlayAttrs(&gOverlays[ii]);
                ret = TRUE;
                break;
            }
            else
            {
                printf("ERROR: second result recvd for overlay %d, template handle == %ld\n",
                       ii, gOverlays[ii].overlaySnapshot);
                break;
            }
        }
    }

    if (ret != TRUE)
    {
        printf("ERROR: non-matching overlay template handle in Add Overlay Completion Event:
[%s]\n",
               presult->hOverlayTemplate);
    }
    return ret;
}


/*****************************************************************************
 * REMOVES THE OVERLAYS WITH A SEPARATE CALL TO THE SM LIBRARAY FOR EACH OVERLAY.
 *****************************************************************************/
int removeOverlaysSeparately()
{
    int ret = TRUE;

    for (int ii = 0; ii < OVERLAYCOUNT; ii++)
    {
        if(gOverlays[ii].overlaySnapshot != MTK_ERROR)
        {
            ret = removeOverlay(gOverlays[ii].overlaySnapshot) && ret;
        }
    }

    return ret;
```

```
}


/*********************************************************************
 * REMOVES THE INDEXED OVERLAY SNAPSHOT.
 *********************************************************************/
int removeOverlay(OB_OVERLAY_HANDLE hOverlay)
{
    SM_REMOVE_OVERLAY_LIST overlayList;
    INIT_SM_REMOVE_OVERLAY_LIST(&overlayList);

    /*********************************************************************
     * SET THE REMOVE OVERLAYS LIST...
     *********************************************************************/
    overlayList.unCount = 1;
    overlayList.overlayHandles[0] = hOverlay;

    printf("Calling sm_RemoveOverlays for overlay handle[%ld]\n", hOverlay);

    /*********************************************************************
     * ... AND CALL REMOVE OVERLAY.
     *********************************************************************/
    if (sm_RemoveOverlays(ghMMDev, &overlayList, NULL) == MTK_ERROR)
    {
        printf("ERROR: returned from sm_RemoveOverlays\n");
        outputLastMTKError();
        releaseHandles();
        return FALSE;
    }

    printf("Waiting for remove overlay completion event...\n");

    /*********************************************************************
     * WAIT FOR THE REMOVE EVENT...
     *********************************************************************/
    long evttype;
    void * evtdatap = NULL;

    if (waitForSMEvent(ghMMDev, SMEV_REMOVE_OVERLAY, &evttype, &evtdatap))
    {
        /*********************************************************************
         * ... AND PROCESS THE SUCCESSFUL EVENT.
         *********************************************************************/
        return handleSMRemoveOverlay(ghMMDev, evttype, evtdatap);
    }
    else
    {
        /*********************************************************************
         * ... OR THE FAILURE.
         *********************************************************************/
        printf("sm_RemoveOverlays() failure on handle [%d] for overlay handle [%d]\n",
               ghMMDev, hOverlay);
        handleSMRemoveOverlay(ghMMDev, evttype, evtdatap);
        return FALSE;
    }
}


/*********************************************************************
 * PROCESS SM LIBRARY REMOVE OVERLAY RELATED EVENTS.
 *********************************************************************/
int handleSMRemoveOverlay(SRL_DEVICE_HANDLE hDev, long evttype, void * evtdatap)
{
    int ret = FALSE;

    switch(evttype)
    {
```

```
    case SMEV_REMOVE_OVERLAY:
        if (evtdatap)
        {
            /*********************************************************************
             * CHECK THAT THE COUNT IS ONE.
             *********************************************************************/
            PSM_REMOVE_OVERLAY_LIST premoveOverlay = (PSM_REMOVE_OVERLAY_LIST)evtdatap;
            if (premoveOverlay->unCount != 1)
            {
                printf("ERROR: remove overlay count not equal to one: value[%d]\n",
                       premoveOverlay->unCount);
            }
            else
            {
                OB_OVERLAY_HANDLE *phandles = premoveOverlay->overlayHandles;
                printf("Removed overlay with handle %d\n", *phandles);
                ret = TRUE;
            }
        }
        else
        {
            printf("ERROR: NULL event data pointer for SMEV_REMOVE_OVERLAY event\n");
        }
        break;

    case SMEV_REMOVE_OVERLAY_FAIL:
        printf("ERROR: SMEV_REMOVE_OVERLAY_FAIL event\n");
        break;

    default:
        printf("ERROR: non-SM event received\n");
    };

    return ret;
}


/*********************************************************************
 * REMOVE ALL OVERLAYS ON MM DEVICE ONE.
 *********************************************************************/
int removeAllOverlays()
{
    /*********************************************************************
     * CALL REMOVE ALL OVERLAYS IN THE SM LIBRARY.
     *********************************************************************/
    printf("Calling sm_RemoveAllOverlays\n");
    if (sm_RemoveAllOverlays(ghMMDev, NULL) == MTK_ERROR)
    {
        printf("ERROR: returned from sm_RemoveAllOverlays\n");
        outputLastMTKError();
        releaseHandles();
        return FALSE;
    }

    printf("Waiting for remove all overlay completion event...\n");

    long evttype;
    void *evtdatap = NULL;

    /*********************************************************************
     * WAIT FOR THE REMOVE ALL OVERLAYS EVENT...
     *********************************************************************/
    if (waitForSMEvent(ghMMDev, SMEV_REMOVE_ALL_OVERLAYS, &evttype, &evtdatap))
    {
        /*****************************************************************
         * ... AND HANDLE THE SUCCESS...
         *****************************************************************/
```

```
        return handleSMRemoveAllOverlays(ghMMDev, evttype, evtdatap);
    }
    else
    {
        /*********************************************************************
         * ... OR THE FAILURE.
         *********************************************************************/
        printf("sm_RemoveAllOverlays() failure on handle [%d]\n", ghMMDev);
        handleSMRemoveAllOverlays(ghMMDev, evttype, evtdatap);
        return FALSE;
    }
}


/*****************************************************************************
 * HANDLE SM LIBRARY REMOVE ALL OVERLAY RELATED EVENTS.
 *****************************************************************************/
int handleSMRemoveAllOverlays(SRL_DEVICE_HANDLE hDev, long evttype, void * evtdatap)
{
    int ret = FALSE;

    switch(evttype)
    {
    case SMEV_REMOVE_ALL_OVERLAYS:
        ret = TRUE;
        break;

    case SMEV_REMOVE_ALL_OVERLAYS_FAIL:
        printf("ERROR: SMEV_REMOVE_ALL_OVERLAYS_FAIL event\n");
        break;

    default:
        printf("ERROR: non-SM event received\n");
    };

    return ret;
}


/*********************************************************************
 * TRANSLATES THE ADD OVERLAY RESULT TO A STRING.
 *********************************************************************/
const char* translateOverlayResult(eSM_ADD_OVERLAY_RESULT result)
{
    static const char * cpadd       = "eSM_ADD_OVERLAY_RESULT_ADD";
    static const char * cpmodify    = "eSM_ADD_OVERLAY_RESULT_MODIFY";
    static const char * cpfail      = "eSM_ADD_OVERLAY_RESULT_FAIL";
    static const char * cperror     = "ERROR - invalid add overlay result";

    switch(result)
    {
    case eSM_ADD_OVERLAY_RESULT_ADD:
        return cpadd;

    case eSM_ADD_OVERLAY_RESULT_MODIFY:
        return cpmodify;

    case eSM_ADD_OVERLAY_RESULT_FAIL:
        return cpfail;

    default:
        return cperror;
    };
}
```

Figure 8. Stream Manipulation Media Toolkit Example Code Output A

Following is the output from running the example code with ADD_OVERLAYS_SEPARATELY and REMOVE_OVERLAYS_SEPARATELY both undefined.

This causes the application to make one call to **sm_AddOverlays( )** to add all the overlays to the multimedia device, and one call to **sm_RemoveAllOverlays( )** to remove all the overlay snapshots from the multimedia device.

```
Stream Manipulation Media Toolkit Example Code Output A
=======================================================

Received media YUV image handle: 1. Setting yuv attributes.
Getting yuv height and width and comparing.
Received yuv media file handle: 4001
Received media JPEG image handle: 2.
Received jpeg media file handle: 4002
Creating frame template
Setting frame [12001] position:
     x: [10.000000]; y: [15.000000]; pos enum[1]
Setting frame [12001] size:
     w: [25.000000]; h: [20.000000]; size enum[1]
Received overlay handle [16001] for overlay index [0]
Setting overlay duration
Setting bounding frame for the overlay template
Setting overlay fill style
Creating frame template
Setting frame [12003] position:
     x: [3.000000]; y: [3.000000]; pos enum[2]
Setting frame [12003] size:
     w: [40.000000]; h: [40.000000]; size enum[1]
Received overlay handle [16002] for overlay index [1]
Setting overlay duration
Setting bounding frame for the overlay template
Setting overlay fill style
Opening mm device 1
MMEV_OPEN event; received handle [1]
Opening mm device 2
MMEV_OPEN event; received handle [3]
Calling getPortInfo() for mm device...
Calling dev_GetTransmitPortInfo() for device handle [1]
Received expected DM event: DMEV_GET_TX_PORT_INFO
Event length matches expected
Calling dev_GetReceivePortInfo() for device handle [1]
Received expected DM event: DMEV_GET_RX_PORT_INFO
Event length matches expected
Calling getPortInfo() for mm2 device...
Calling dev_GetTransmitPortInfo() for device handle [3]
Received expected DM event: DMEV_GET_TX_PORT_INFO
Event length matches expected
Calling dev_GetReceivePortInfo() for device handle [3]
Received expected DM event: DMEV_GET_RX_PORT_INFO
Event length matches expected
Calling connectPorts() for mm tx and mm2 rx ports...
Calling createConnectInfoList() for dev handle [1]...
Transmit port list count: 2
Transmission port list type_tx: 1
Receive port list count: 2
Receive port list type_rx: 1
Transmission port list type_tx: 2
Receive port list count: 2
Receive port list type_rx: 1
Receive port list type_rx: 2
Calling dev_PortConnect() for dev handle [1]...
```

```
Received expected DM event: DMEV_PORT_CONNECT
Calling connectPorts() for mm2 tx and mm rx ports...
Calling createConnectInfoList() for dev handle [3]...
Transmit port list count: 2
Transmission port list type_tx: 1
Receive port list count: 2
Receive port list type_rx: 1
Transmission port list type_tx: 2
Receive port list count: 2
Receive port list type_rx: 1
Receive port list type_rx: 2
Calling dev_PortConnect() for dev handle [3]...
Received expected DM event: DMEV_PORT_CONNECT
Calling sm_AddOverlay for 2 overlays
Waiting for add overlay completion event...
Event length matches expected
Received expected event [SMEV_ADD_OVERLAY]
successful event user context match
Received overlay: template handle [16001]; snapshot handle [16003];
result[eSM_ADD_OVERLAY_RESULT_ADD]
Checking bounding frame position and size for the overlay snapshot against set values
Snapshot overlay position information matches settings.
Snapshot overlay size information matches settings.
Checking overlay fill style
Snapshot overlay fill style matches set value.
Received overlay: template handle [16002]; snapshot handle [16004];
result[eSM_ADD_OVERLAY_RESULT_ADD]
Checking bounding frame position and size for the overlay snapshot against set values
Snapshot overlay position information matches settings.
Snapshot overlay size information matches settings.
Checking overlay fill style
Snapshot overlay fill style matches set value.
Calling sm_RemoveAllOverlays
Waiting for remove all overlay completion event...
Received expected event [SMEV_REMOVE_ALL_OVERLAYS]
Successfully destroyed YUV media template
Successfully destroyed media template
Successfully destroyed yuv image
Successfully destroyed jpeg image
Successfully destroyed frame template[12001]
Successfully destroyed frame template[12003]
Successfully destroyed overlay [16001]
Successfully destroyed overlay [16002]
Successfully closed mm device
Successfully closed mm2 device
```

Figure 9. Stream Manipulation Media Toolkit Example Code Output B

Following is the output from running the example code with ADD_OVERLAYS_SEPARATELY and REMOVE_OVERLAYS_SEPARATELY defined.

This causes the application to make a separate call to **sm_AddOverlays( )** to add each overlay template to the multimedia device, and a separate call to **sm_RemoveOverlays( )** to remove each overlay snapshot from the multimedia device.

```
Stream Manipulation Media Toolkit Example Code Output B
=======================================================

Received media YUV image handle: 1. Setting yuv attributes.
Getting yuv height and width and comparing.
Received yuv media file handle: 4001
Received media JPEG image handle: 2.
Received jpeg media file handle: 4002
Creating frame template
Setting frame [12001] position:
     x: [10.000000]; y: [15.000000]; pos enum[1]
Setting frame [12001] size:
     w: [25.000000]; h: [20.000000]; size enum[1]
Received overlay handle [16001] for overlay index [0]
Setting overlay duration
Setting bounding frame for the overlay template
Setting overlay fill style
Creating frame template
Setting frame [12003] position:
     x: [3.000000]; y: [3.000000]; pos enum[2]
Setting frame [12003] size:
     w: [40.000000]; h: [40.000000]; size enum[1]
Received overlay handle [16002] for overlay index [1]
Setting overlay duration
Setting bounding frame for the overlay template
Setting overlay fill style
Opening mm device 1
MMEV_OPEN event; received handle [1]
Opening mm device 2
MMEV_OPEN event; received handle [3]
Calling getPortInfo() for mm device...
Calling dev_GetTransmitPortInfo() for device handle [1]
Received expected DM event: DMEV_GET_TX_PORT_INFO
Event length matches expected
Calling dev_GetReceivePortInfo() for device handle [1]
Received expected DM event: DMEV_GET_RX_PORT_INFO
Event length matches expected
Calling getPortInfo() for mm2 device...
Calling dev_GetTransmitPortInfo() for device handle [3]
Received expected DM event: DMEV_GET_TX_PORT_INFO
Event length matches expected
Calling dev_GetReceivePortInfo() for device handle [3]
Received expected DM event: DMEV_GET_RX_PORT_INFO
Event length matches expected
Calling connectPorts() for mm tx and mm2 rx ports...
Calling createConnectInfoList() for dev handle [1]...
Transmit port list count: 2
Transmission port list type_tx: 1
Receive port list count: 2
Receive port list type_rx: 1
Transmission port list type_tx: 2
Receive port list count: 2
Receive port list type_rx: 1
Receive port list type_rx: 2
Calling dev_PortConnect() for dev handle [1]...
Received expected DM event: DMEV_PORT_CONNECT
Calling connectPorts() for mm2 tx and mm rx ports...
Calling createConnectInfoList() for dev handle [3]...
Transmit port list count: 2
Transmission port list type_tx: 1
Receive port list count: 2
Receive port list type_rx: 1
Transmission port list type_tx: 2
Receive port list count: 2
Receive port list type_rx: 1
Receive port list type_rx: 2
Calling dev_PortConnect() for dev handle [3]...
```

```
Received expected DM event: DMEV_PORT_CONNECT
Calling sm_AddOverlay for overlay handle[16001]
Waiting for add overlay completion event...
Event length matches expected
Received expected event [SMEV_ADD_OVERLAY]
successful event user context match
Received overlay: template handle [16001]; snapshot handle [16003];
result[eSM_ADD_OVERLAY_RESULT_ADD]
Checking bounding frame position and size for the overlay snapshot against set values
Snapshot overlay position information matches settings.
Snapshot overlay size information matches settings.
Checking overlay fill style
Snapshot overlay fill style matches set value.
Calling sm_AddOverlay for overlay handle[16002]
Waiting for add overlay completion event...
Event length matches expected
Received expected event [SMEV_ADD_OVERLAY]
successful event user context match
Received overlay: template handle [16002]; snapshot handle [16004];
result[eSM_ADD_OVERLAY_RESULT_ADD]
Checking bounding frame position and size for the overlay snapshot against set values
Snapshot overlay position information matches settings.
Snapshot overlay size information matches settings.
Checking overlay fill style
Snapshot overlay fill style matches set value.
Calling sm_RemoveOverlays for overlay handle[16003]
Waiting for remove overlay completion event...
Received expected event [SMEV_REMOVE_OVERLAY]
Removed overlay with handle 16003
Calling sm_RemoveOverlays for overlay handle[16004]
Waiting for remove overlay completion event...
Received expected event [SMEV_REMOVE_OVERLAY]
Removed overlay with handle 16004
Successfully destroyed YUV media template
Successfully destroyed media template
Successfully destroyed yuv image
Successfully destroyed jpeg image
Successfully destroyed frame template[12001]
Successfully destroyed frame template[12003]
Successfully destroyed overlay [16001]
Successfully destroyed overlay [16002]
Successfully closed mm device
Successfully closed mm2 device
```

Glossary

active talker: A participant in a conference who is providing “non-silence” energy.

automatic gain control (AGC): An electronic circuit used to maintain the audio signal volume at a constant level. AGC maintains nearly constant gain during voice signals, thereby avoiding distortion, and optimizes the perceptual quality of voice signals by using a new method to process silence intervals (background noise).

asynchronous function: A function that allows program execution to continue without waiting for a task to complete. To implement an asynchronous function, an application-defined event handler must be enabled to trap and process the completed event. Contrast with synchronous function.

bit mask: A pattern which selects or ignores specific bits in a bit-mapped control or status field.

bitmap: An entity of data (byte or word) in which individual bits contain independent control or status information.

board device: A board-level object that maps to a virtual board.

buffer: A block of memory or temporary storage device that holds data until it can be processed. It is used to compensate for the difference in the rate of the flow of information (or time occurrence of events) when transmitting data from one device to another.

bus: An electronic path that allows communication between multiple points or devices in a system.

busy device: A device that has one of the following characteristics: is stopped, being configured, has a multitasking or non-multitasking function active on it, or I/O function active on it.

channel device: A channel-level object that can be manipulated by a physical library, such as an individual telephone line connection. A channel is also a subdevice of a board.

conference: Ability for three or more participants in a call to communicate with one another in the same call.

configuration file: An unformatted ASCII file that stores device initialization information for an application.

CT Bus: Computer Telephony bus. A time division multiplexing communications bus that provides 4096 time slots for transmission of digital information between CT Bus products. See TDM bus.

data structure: Programming term for a data element consisting of fields, where each field may have a different type definition and length. A group of data structure elements usually share a common purpose or functionality.

device: A computer peripheral or component controlled through a software device driver. A Dialogic® voice and/or network interface expansion board is considered a physical board containing one or more logical board devices, and each channel or time slot on the board is a device.

device channel: A voice data path that processes one incoming or outgoing call at a time (equivalent to the terminal equipment terminating a phone line).

device driver: Software that acts as an interface between an application and hardware devices.

device handle: Numerical reference to a device, obtained when a device is opened using **xx_open( )**, where *xx* is the prefix defining the device to be opened. The device handle is used for all operations on that device.

device name: Literal reference to a device, used to gain access to the device via an **xx_open( )** function, where *xx* is the prefix defining the device to be opened.

driver: A software module which provides a defined interface between a program and the firmware interface.

DTMF (Dual-Tone Multifrequency): Push-button or touch-tone dialing based on transmitting a high- and a low-frequency tone to identify each digit on a telephone keypad.

extended attribute functions: A class of functions that take one input parameter and return device-specific information. For instance, a voice device's extended attribute function returns information specific to the voice devices. Extended attribute function names are case-sensitive and must be in capital letters. See also standard runtime library (SRL).

firmware: A set of program instructions that reside on an expansion board.

idle device: A device that has no functions active on it.

region: A frame within the root; corresponds to the visual display of a single video stream. The root may contain a single region or multiple regions. The root may be divided into regions where no region overlaps, or regions may overlap.

root (root frame): The whole video frame or picture that is viewed by participants of a multimedia conference.

RFU: Reserved for future use.

route: Assign a resource to a time slot.

SRL: See **Standard Runtime Library**.

standard attribute functions: Class of functions that take one input parameter (a valid device handle) and return generic information about the device. For instance, standard attribute functions return IRQ and error information for all device types. Standard attribute function names are case-sensitive and must be in capital letters. Standard attribute functions for all Dialogic® devices are contained in the SRL. See standard runtime library (SRL).

standard runtime library (SRL): A Dialogic® software resource containing event management and standard attribute functions and data structures used by all Dialogic® devices, but which return data unique to the device.

synchronous function: Blocks program execution until a value is returned by the device. Also called a blocking function. Contrast with asynchronous function.

TDM (Time Division Multiplexing): A technique for transmitting multiple voice, data, or video signals simultaneously over the same transmission medium. TDM is a digital technique that interleaves groups of bits from each signal, one after another. Each group is assigned its own "time slot" and can be identified and extracted at the receiving end. See also time slot.

TDM bus: Time division multiplexing bus. A resource sharing bus such as the SCbus or CT Bus that allows information to be transmitted and received among resources over multiple data lines.

termination condition: An event or condition which, when present, causes a process to stop.

termination event: An event that is generated when an asynchronous function terminates. See also asynchronous function.

thread (Windows®): The executable instructions stored in the address space of a process that the operating system actually executes. All processes have at least one thread, but no thread belongs to more than one process. A multithreaded process has more than one thread that are executed seemingly simultaneously. When the last thread finishes its task, then the process terminates. The main thread is also referred to as a primary thread; both main and primary thread refer to the first thread started in a process. A thread of execution is just a synonym for thread.

time division multiplexing (TDM): See TDM (Time Division Multiplexing).

time slot: The smallest, switchable data unit on a TDM bus. A time slot consists of 8 consecutive bits of data. One time slot is equivalent to a data path with a bandwidth of 64 kbps. In a digital telephony environment, a normally continuous and individual communication (for example, someone speaking on a telephone) is (1) digitized, (2) broken up into pieces consisting of a fixed number of bits, (3) combined with pieces of other individual communications in a regularly repeating, timed sequence (multiplexed), and (4) transmitted serially over a single telephone line. The process happens at such a fast rate that, once the pieces are sorted out and put back together again at the receiving end, the speech is normal and continuous. Each individual, pieced-together communication is called a time slot.

video layout: Defines how contributing video streams are mixed and visually arranged for viewing by participants of a multimedia conference. The layout can also include visual attributes such as outlines, borders, and image overlays.