Book 8—Graphics Libraries

TriMedia

Version 2.0 beta

Table of Contents

Chapter 1 Introduction to the Graphics Libraries

Chapter 2 2D Graphics API

2D Graphics Library Overview 16
Rectangle Coordinates Specification 16
Supported Buffer Types 16
Graphics and Video Images Blending Specification in the DTV Buffer Types .. 17
Drawing Primitives APIs 18
Clipping 19
Drawing Rules 19
Fonts: TMFont and TMFont2 19
TMFont 19
Font TM Font Files 19
TMFont2 20
TMFont2 Font Files 21

How to Use the 2D Graphics Library 22
Necessary Items 22
Programs that use 2D Graphics Library 22
How to Load Fonts 22
PC Host 22
stand-alone 22
Technical Difficulties with 2D Graphics Library 23

Returned Error Messages 23

2D API Data Structures 24
tsa2DCapabilities_t 26
tsaYUVAColor_t 26
tsaYUVColor_t 27
tsaRGBColor_t 27
tsa2DColorType_t 28
tsa2DColor_t 28
tsaYUVA4Color_t 29
tsa2DIndexColorLUT_t 30

tsa2DCoordinate_t 30
tsa2DRect_t 31
tsa2DImageType_t 31
tsa2DImage_t 32
tsa2DTextStyle_t 33
tsa2DFontInfoFlag_t 33
tsaFontTMCharMetrics_t 34
tsaFontTM_t 35
tsaTMFont2CharMetrics 36
tsaTMFont2 37
tsa2DFontType_t 38
tsa2DFont_t 38
tsa2DContext_t 39

2D API Functions 40
tsa2DGetCapabilities 42
tsa2DOpen 43
tsa2DClose 43
tsa2DRGBtoYUV 44
tsa2DYUVtoRGB 45
tsa2DLoadIndexColorLUT 46
tsa2DUnLoadIndexColorLUT 47
tsa2DGetColorFmIndex 48
tsa2DPointNC 49
tsa2DLineNC 50
tsa2DFillRectNC 51
tsa2DImageNC 52
tsa2DTextNC 53
tsa2DSetPixel 55
tsa2DGetPixel 56
tsa2DPoint 57
tsa2DLine 58
tsa2DFillRect 59
tsa2DFillPoly 60
tsa2DImage 61
tsa2DText 62
tsa2DBlt 64
tsa2DBltRegion 65
tsa2DPolyPoint 66
tsa2DPolyLine 67
tsa2DPolyFillRect 68
tsa2DPolyImage 69

tsa2DPolyText 70
tsa2DPolyBlt 72
tsa2DGetStrWidth 73
tsa2DGetFontInfo 74
tsa2DTMFontSetCharSpacingInString 75
tsa2DTMFontGetCharSpacingInString 76
tsa2DLoadFont 77
tsa2DUnLoadFont 78

Chapter 3 Closed-Captioning (EIA-608) API

DTVCC Decoder (EIA-608) Overview 80

Operation 81

Sample Application 82

VrendEia608 API Functions 83
tmalVrendEia608Open 84
tmolVrendEia608Open 85
tmalVrendEia608Close 86
tmolVrendEia608Close 86
tmalVrendEia608Start 87
tmolVrendEia608Start 87
tmalVrendEia608Stop 88
tmolVrendEia608Stop 88
tmalVrendEia608GetCapabilities 89
tmolVrendEia608GetCapabilities 89
tmolVrendEia608GetInstanceSetup 90
tmalVrendEia608InstanceConfig 90
tmolVrendEia608InstanceConfig 91
tmalVrendEia608InstanceSetup 92
tmolVrendEia608InstanceSetup 92
tmalVrendEia608RedrawFunc 93
tmalVrendEia608DecodePacket 94
tmalVrendEia608FieldVsync 95

VrendEia608 API Enumerations and Data Structures 96
Eia608_Field_t 97
Eia608_Service_t 98
Eia608_XDSPackTypes_t 99
tmalVrendEia608ConfigTypes_t 102
tmalVrendEia608InstanceSetup_t 103
tmolVrendEia608InstanceSetup_t 105

Eia608_ATVEFPackTypes_t 107
tmVrendEia608ProgressVCHIP_t 108
tmVrendEia608ProgressXDS_t 109
tmVrendEia608ProgressATVEF_t 110

Chapter 4 Closed-Captioning (EIA-708) API

DTVCC Decoder (EIA-708) Overview 112
Background 112

DTVCC Decoder (EIA-708) Inputs and Outputs 112
Compliance With the DTVCC Standard 113
Multiple Service Channel Decoding 113

DTVCC Decoder (EIA-708) Progress 113

DTVCC Decoder (EIA-708) Error 114
Error codes 114

DTVCC Decoder (EIA-708) API Data Structures 114
tmolVrendEia708Capabilities_t 115
tmalVrendEia708Capabilities_t 115
tmalVrendEia708InstanceSetup_t 116
tmolVrendEia708InstanceSetup_t 116
tmVrendEia708Fonts_t 118
tmVrendEia708FontStyles_t 119
tmVrendEia708AR_t 120
tmVrendEia708ServDecSetup_t 120
tmVrendEia708ConfigCommands_t 121
tmVrendEia708ConfigParams_t 124

DTVCC Decoder (EIA-708) API Functions 125
tmolVrendEia708GetCapabilities 126
tmolVrendEia708Open 127
tmolVrendEia708Close 127
tmolVrendEia708GetInstanceSetup 128
tmolVrendEia708InstanceSetup 129
tmolVrendEia708Start 130
tmolVrendEia708Stop 130
tmolVrendEia708InstanceConfig 131
tmolVrendEia708FieldVsync 132

Chapter 5 HTML Parser (HtmlParser) API

Overview **134**
Modules 134
Header Files 135
Resource Files in the Database 135
HTML Pages 135
TM Fonts 135
Widget Images 136
Other Image Files 137
TriMedia Extensions to the HTML 137
Button in INPUT tag 137
Horizontal Slider 137
How to Use the HTML Parser and HTML Renderer Libraries 138
HTML Renderer Navigation Functions 139
HTML Renderer 'Get Information' Functions 139
Example (exHtml) Overview 139
Wrapper Function: myGetObject 140

HTML Data Structures **141**
tsaHtmlFont_t 142
tsaHtmlWidgetStateGeneric_t 142
tsaHtmlWidgetStateTextline_t 143
tsaHtmlWidgetStatePassword_t 144
tsaHtmlWidgetStateRadio_t 145
tsaHtmlWidgetStateCheckbox_t 146
tsaHtmlWidgetStateButton_t 147
tsaHtmlWidgetStateSubmit_t 148
tsaHtmlWidgetStateReset_t 149
tsaHtmlWidgetStateImage_t 150
tsaHtmlWidgetStateFile_t 151
tsaHtmlWidgetStateHidden_t 152
tsaHtmlWidgetStateSelect_t 153
tsaHtmlWidgetStateTextarea_t 155
tsaHtmlWidgetStateSlider_t 156

HTML Enumerated Types **157**
tsaHtmlHotspotType_t 158
tsaHtmlFontStyle_t 159
tsaHtmlImageAlign_t 159

HTML API Data Structures 160
tsaHtmlParserCapabilities_t 161
tsaHtmlParserInstanceSetup_t 162
tsaHtmlParserFrameState_t 163
tsaHtmlParserSetupFlags_t 164

HTML API Functions 165
tsaHtmlParserGetCapabilities 166
tsaHtmlParserOpen 167
tsaHtmlParserGetInstanceSetup 168
tsaHtmlParserInstanceSetup 169
tsaHtmlParserClose 170
tsaHtmlParserLoadUrl 171
tsaHtmlParserLoadHtml 172
tsaHtmlParserUnload 173

HTML Tags Supported 174

Chapter 6 HTML Renderer (HtmlRender) API

Overview 180
Modules 180
Header Files 180
The TriMedia HTML Parser (HtmlParser) 180

HTML Renderer API Data Structures 181
tsaHtmlRenderCapabilities_t 182
tsaHtmlRenderInstanceSetup_t 183
tsaHtmlRenderWidgetState_t 185
tsaHtmlRenderSetupFlags_t 186
tsaHtmlRenderHotspotDir_t 186
tsaHtmlRenderScrollDir_t 187

HTML Renderer API Functions 188
tsaHtmlRenderGetCapabilities 189
tsaHtmlRenderOpen 190
tsaHtmlRenderGetInstanceSetup 191
tsaHtmlRenderInstanceSetup 192
tsaHtmlRenderClose 193
tsaHtmlRenderFrameStateCreate 194
tsaHtmlRenderFrameStateDestroy 195
tsaHtmlRenderRenderFrame 196
tsaHtmlRenderRenderAllFrames 197
tsaHtmlRenderRenderHotspot 198

tsaHtmlRenderGetFrameId....199
tsaHtmlRenderGetCurrentHotspot....200
tsaHtmlRenderGetHotspot....201
tsaHtmlRenderGetNumHotspots....202
tsaHtmlRenderGetSubFrame....203
tsaHtmlRenderGetNumSubFrames....204
tsaHtmlRenderFollowNamedLink....205
tsaHtmlRenderScrollScreen....206

Chapter 7 Object Manager (OM) API

Object Manager Overview....208
Object Manager....208
Object Manager Database Builder....208
Database Builder....209
Database Loader....209
Database Format....210

Object Manager API Data Structures....212
tsaOMCapabilities_t....213
tsaOMInstanceSetup_t....213
tsaOMHTML_t....214

Object Manager API Enumerated Types....215
tsaOMType_t....216

Object Manager API Functions....217
tsaOMGetCapabilities....218
tsaOMOpen....218
tsaOMGetInstanceSetup....219
tsaOMInstanceSetup....220
tsaOMClose....221
tsaOMGetObject....222

Chapter 8 Widget API

Introduction....224
Widget Library Overview....224
Basic Operations....225
How to Create a Standard Widget....225
Widget Example Programs (exWidget) Overview....226
Wrapper Function: myGetObject....226

Widget Library Data Structures **227**
tsaWidgetCapabilities_t 228
tsaWidgetInstanceSetup_t 229
tsaWidgetInstVar_t 230
_tsaWidgetObject_t 231

Widget Library Functions **232**
tsaWidgetGetCapabilities 233
tsaWidgetOpen 234
tsaWidgetGetInstanceSetup 235
tsaWidgetInstanceSetup 236
tsaWidgetClose 237

Standard Widget Set **238**

Standard Widget Set Enumerated Types **239**
tsaWidgetButtonIndex_t 240
tsaWidgetImageIndex_t 241
tsaWidgetPasswordIndex_t 242
tsaWidgetSelectIndex_t 243
tsaWidgetSliderIndex_t 245
tsaWidgetTextareaIndex_t 246
tsaWidgetTextlineIndex_t 248
tsaWidgetToggleIndex_t 249

Standard Widget Set Functions and Macros **250**
tsaWidgetCreateButton 252
tsaWidgetCreateImage 253
tsaWidgetCreatePassword 254
tsaWidgetCreateSelect 255
tsaWidgetCreateSlider 256
tsaWidgetCreateTextarea 257
tsaWidgetCreateTextline 258
tsaWidgetCreateToggle 259
tsaWidgetPlot 260
tsaWidgetPLOT 261
tsaWidgetUpdate 262
tsaWidgetUPDATE 263
tsaWidgetGet 264
tsaWidgetGET 265
tsaWidgetSet 266
tsaWidgetSET 267
tsaWidgetGetPacket 268

tsaWidgetSetPacket 268
tsaWidgetGetX 269
tsaWidgetSetX 269
tsaWidgetGetY 270
tsaWidgetSetY 270
tsaWidgetGetWidth 271
tsaWidgetSetWidth 271
tsaWidgetGetHeight 272
tsaWidgetSetHeight 272
tsaWidgetGetuserData 273
tsaWidgetSetuserData 273
tsaWidgetFill 274
tsaWidgetDestroy 275

How to Write Widgets 276
Widget Library Framework 276
Widget Example (WidgetTemplate) Overview 277
WidgetTextBox.h 277
WidgetTextBoxInternal.h 278
WidgetTextBox.c and WidgetTextBox2.c 278
tsaWidgetCreateTextBox 278
TextBoxGet 278
TextBoxSet 279
TextBoxPlot 279

Chapter 9 Window Manager (WM) API

Introduction 282

Windows 282
Window Types 282
Instances 282
Video Out 283
Redrawing 283
Moving 283
Stacking Order 283
Display and Hiding 284
Scrolling 284
Locking by User 284
Reentrancy 284
Parent Windows 285

Returned Error Messages 286

Window Manager API Data Structures 287
tsaWMStackingOrder_t 288
ptsaRedrawCallbackFun_t 289
tsaWMCapabilities_t 290
tsaWMInstanceSetup_t 290

Window Manager API Functions 291
tsaWMGetCapabilities 292
tsaWMOpen 292
tsaWMClose 293
tsaWMInstanceSetup 293
tsaWMCreateRealWindow 294
tsaWMCreateVirtualWindow 295
tsaWMDestroyWindow 296
tsaWMMoveWindow 297
tsaWMRaiseWindow 298
tsaWMLowerWindow 299
tsaWMRaiseAllWindows 300
tsaWMLowerAllWindows 300
tsaWMDisplayWindow 301
tsaWMHideWindow 302
tsaWMRedrawWindow 303
tsaWMChangeViewingWindow 304
tsaWMLockWindow 305
tsaWMUnlockWindow 306

Chapter 1

Introduction to the Graphics Libraries

Graphics support on the Trimedia is provided by eight libraries that implement such services as HTML browsing, close captioning, GUI building blocks, and low-level graphics primitives. Graphics libraries are layered, from the 2D Graphics Library and the Object Manager at the lowest level to higher level services like close captioning and rendering HTML pages. All libraries comply with the Trimedia Software Architecture (TSA).

The 2D Graphics library implements low-level primitives such as bitblts (bit block transfers), polygons, points, line, image drawing, and filled rectangles. The Object Manager (OM) provides a way to create a database of HTML pages, images and fonts. These two components are at the lowest level, in the sense that all other graphics components use one or both of these to implement their own services.

At the next level, there are three components, dealing with more abstract primitives. The Window Manager (WM) allows independent windows to be composited together. The WM is dependent on the 2D library. The Widget library provides primitives, such as buttons and sliders, that can be used to build user interfaces, and is dependent on both the 2D and OM libraries. Finally the HTML Parser enables an application to parse HTML pages into a format that can be rendered by the HTML Renderer. The HTML Parser also uses the 2D and OM libraries internally.

The HTML Renderer and Close-Captioning APIs exist at the highest level. The HTML Renderer processes the output of the HTML Parser and uses the 2D, OM, Widget, and WM libraries to render HTML pages to the screen. Two close-captioning APIs exist, one to support the Eia608 standard and another to support the DTVCC (Eia708) standard. Both use the WM and 2D libraries to integrate close-captioning with video and other onscreen graphics, and both are TSSA-compliant.

Chapter 2

2D Graphics API

Topic	Page
2D Graphics Library Overview	16
How to Use the 2D Graphics Library	22
Returned Error Messages	23
2D API Data Structures	24
2D API Functions	40

Note
This component library is not included with the basic TriMedia SDE, but is available as a part of other software packages, under a separate licensing agreement. Please visit our web site (www.trimedia.philips.com) or contact your TriMedia sales representative for more information.

2D Graphics Library Overview

The 2D Graphics Library draws 2D lines, points, text, rectangles and polygons on a buffer that the user passes in. It is compliant with TriMedia Software Architecture (TSA). The 2D Graphics Library is decoupled from the hardware, hence, it does not have an instance setup function. It renders on the packet buffer passed in from the user. It supports eight buffer types, and they are: YUV422 planar, video-overlay sequence, DTVCM-YUV422 planar, DTVCM-overlay sequence, YUV422 planar with 4-bit alpha, RGB888, RGB565, and RGB555A.

The font renderer renders two font types: TMFont and TMFont2. It provides color conversion between RGB and YUV color spaces. The supported drawing primitives are: Point, Line, Text, Fill Rectangle, Fill Polygon, Image, and Blt.

Rectangle Coordinates Specification

The packet buffer size is derived from the imageWidth and imageHeight fields of the tmVideoFormat_t of the packet buffer. The 2D Graphics Library draws only within the packet boundary. The origin (0,0) of the rectangle is at the top left corner of the buffer. Therefore the upper left coordinate of the tsa2DRect_t structure is defined to be less than or equal, in both X and Y, to the bottom right coordinate. All API functions that draw rectangles generate an upper left and bottom right point from user-specified arguments. Since this kind of min/max box can be derived from any two points, no ordering is assumed for points supplied as arguments to the 2D Graphics Library.

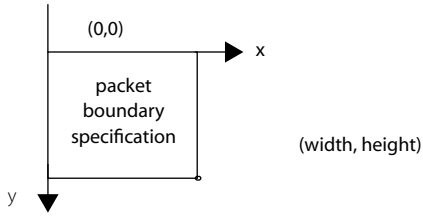


Figure 1 Rectangle Coordinates

Supported Buffer Types

2D Library supports the following buffer types:

- vdfYUV422Planar
- vdfYUV422Sequence
- vdfDTVCMPlanar
- vdfDTVCMSequence
- vdfRGB24

- vdfRGB16
- vdfRGB555A
- vdfYUV422PlanarAlpha4

Buffer type are specified through the **dataSubtype** entry of the video format. For example, to specify **vdfDTVCMPlanar**, use the following code entry shown below:

```
yuvFmt.dataSubtype = vdfYUV422Planar
```

Graphics and Video Images Blending Specification in the DTV Buffer Types

In the DTV environment, the **vdfDTVCMPlanar** and **vdfDTVCMSequence** buffer types are the corresponding YUV422 and overlay buffer types with the consideration of color multiplexing between Graphics and Video.

Blending Graphics and Video Streams

The following flags are used to specify the blending between Graphics and Video:

- **vdfDTVCM_0Video**
- **vdfDTVCM_25Video**
- **vdfDTVCM_50Video**
- **vdfDTVCM_75Video**
- **vdfDTVCM_DontCare**

The blending factor are specified through the description entry of the video format. For example, to specify a blending factor ratio of 25% of video and 75% of graphics, enter the following:

```
yuvFmt.description = vdfDTVCM_25Video; /* 25% Video, 75% graphics */
```

Note
When the graphics buffers are filled with color key values, it displays 100% of Video and 0% of Graphics.

Blending of Anti-Aliased Text and Video Streams

The following two additional flags are used to specify the blending between anti-aliased Text and Video streams:

- **vdfDTVCM_MAP_GAtoVA_W_FC**
- **vdfDTVCM_MAP_GAtoVA_W_FCBC**

vdfDTVCM_MAP_GAtoVA_W_FC maps the encoded alpha blending values (0-15) in the text to the color multiplexor blending values (that is, LSBs of UV: 00, 01, 10, 11) with the foreground color.

vdfDTVCM_MAP_GAtoVA_W_FCBC maps the encoded alpha blending values (0-15) in the text to the color multiplexor blending values (i.e. LSBs of UV: 00, 01, 10, 11) with the resulting color of alpha blended foreground and background colors.

For example:

```
((ptmVideoFormat_t)pYuvPkt->header->format)->description =
                                                vdfDTVCM_MAP_GAtoVA_W_FC;
```

Table 1 Blending Values

Flag	Graphics Alpha Blending Values	Video Alpha Blending Values
vdfDTVCM_MAP_GAtoVA_W_FC	0 to 15	00, 01, 10, 11
vdfDTVCM_MAP_GAtoVA_W_FCBC	0 to 15	00, 01, 10, 11

Drawing Primitives APIs

There are three sets of drawing primitive APIs:

- No Graphics Context APIs
- Poly APIs
- Graphics Context APIs

No Graphics Context APIs

The following drawing primitives API do not use graphics context:tsa2DPointNC, tsa2DLineNC, tsa2DFillRectNC, tsa2DImageNC, and tsa2DTextNC.

Instead, the required information is supplied through input arguments.

Poly APIs

The following poly APIs are: **tsa2DPolyPoints**, **tsa2DPolyLine**, **tsa2DPolyFillRect**, **tsa2D-PolyImage**, **tsa2DPolyText**, and **tsa2DPolyBlt**.

These Poly functions do drawing on multiple packets (i.e. **numPkt**).

Within each packet or each set of packets, they can also draw multiple times (i.e. specify in **pNumPerPkt**).

pPkList is a pointer to an array of packet pointers. The number of packet pointers should equal to **numPkt**. **pPtList** is a pointer to an array of 2D coordinates. The number of coordinates should equal to:

```
(pNumPerPkt[0] + ... + pNumPerPkt[numPkt-1])
```

pColor is a pointer to **tsa2DColor_t**. The entry, **pColor->pColorData**, is a pointer to an array of 2D colors (ex: **tsaYUVColor_t**). The number of colors should be equal to:

```
2D API Data Structure Descriptions(pNumPerPkt[0]+...+pNumPerPkt[numPkt-1])
```


Graphics Context APIs

The following drawing primitive APIs do use graphics context of the input parameter: tsa2DGetPixel, tsa2DSetPixel, tsa2DPoint, tsa2DLine, tsa2DText, tsa2DImage, gsa2DFillRect, tsa2DFillPoly, tsa2DBlt, and tsa2DBltRegion.

Clipping

The 2D Graphics Library supports clipping on all primitives. Only the portion of a primitive falling within the packet boundary, if any, is drawn. The clipping is pixel exact, meaning that the pixels generated for a clipped primitive are a subset of the pixels generated for the unclipped primitive.

Drawing Rules

The 2D Graphics Library uses the 'upper left pixel in, bottom right pixel out' rule when determining which pixels belong to filled rectangle, image, and BitBlt drawing primitives. This means that the bottom row and rightmost column of the primitives mentioned are not drawn. This rule ensures that in the case of adjacent primitives, pixels along shared borders belong to exactly one primitive.

Fonts: TMFont and TMFont2

2D Graphics Library supports two types of fonts, TMFont and TMFont2. They are both bitmap type of fonts with slight variation in the font information data structures.

TMFont

The information of a particular font is stored in (font.mtr and font.bit) files. When **tsa2DLoadFont** is called, it loads the information into library. You need to provide information regarding the path of font files and the library returns a fonID after it loads in the font. **tsa2DUnLoadFont** unloads the font specified in the fontID.

Font TM Font Files

Below is a picture description of the TMFont font files. The .mtr file contains information for the font and each character. The .bit file has character bitmaps information. Figure 3 provides a graphic example.

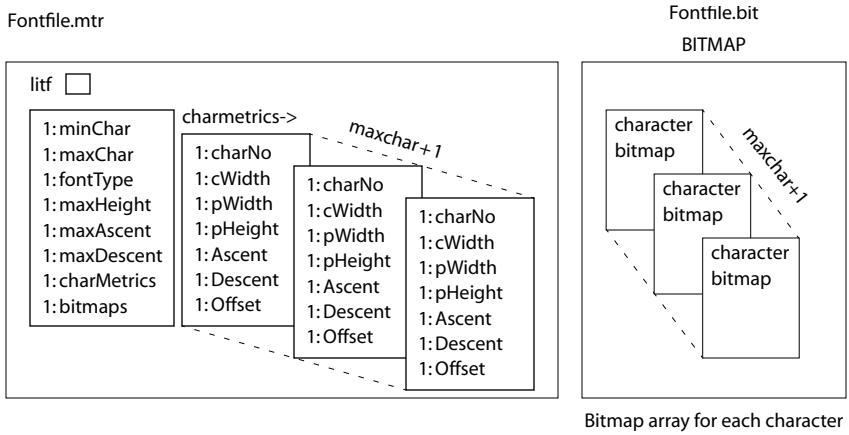


Figure 2 TMFont Font Files

TMFont2

The information of a particular font is stored in (font.tm and font.bit) files. When **tsa2DLoadFont** is called, it loads the information into the 2D library. You need to provide information regarding the path of font files and the library returns a fontID after it loads in the font. **tsa2DUnLoadFont** unloads the font specified in the fontID.

TMFont2 Character Metrics

Each pixel is represented with 4 bits of blending information (i.e. the color blending between text color and background color). 0xF shows the pixel with the text color. 0x0 shows the background color. The values in between are blended proportionally.

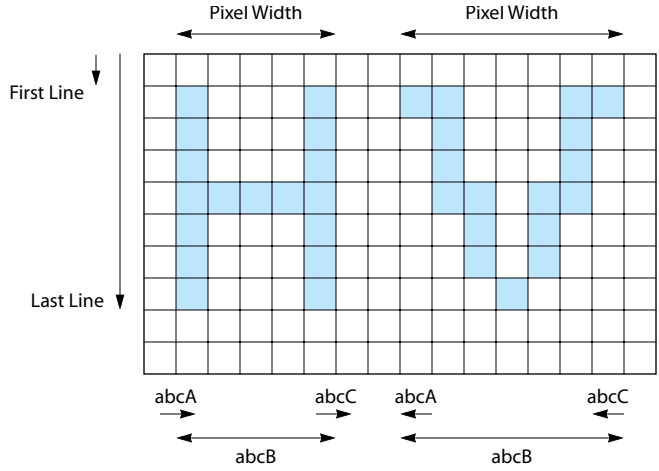


Figure 3 TMFont Character Metric Graphic Representation

TMFont2 Font Files

Below is a picture description of entries in the **tsaTMFont2CharMetrics**:

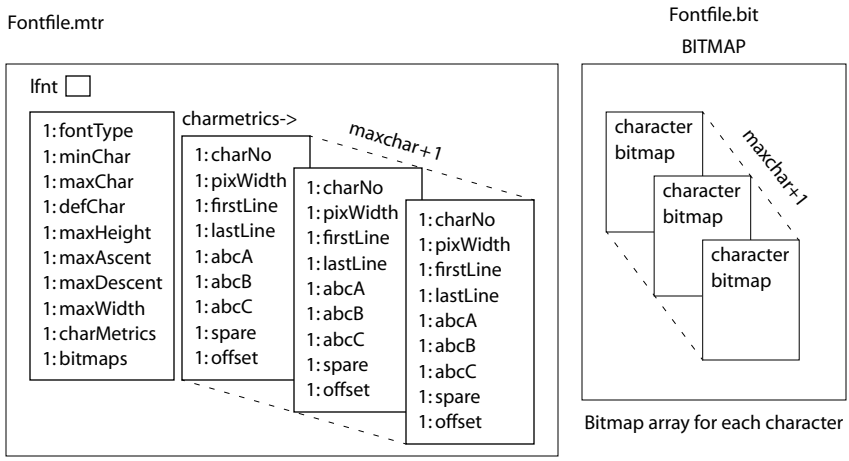


Figure 4 TMFont2 Font Files

How to Use the 2D Graphics Library

To use the 2D Graphics Library, you must use the specified hardware, and programs discussed in this section.

Necessary Items

The following items are necessary in order to use 2D Graphics:

1. TriMedia board with TV.
2. TriMedia Compilation System (TCS).
3. TriMedia Application Software (TAS) and specifically libtm2D.a.
4. Optional: Example program using 2D Library.

Programs that use 2D Graphics Library

An application program needs to get an instance ID from **tsa2DOpen** first, before using: font APIs, drawing APIs, and color conversion APIs. Use **tsa2DClose** when done.

1. Call **tsa2Dopen** to get an instance ID.
2. Use font APIs: **tsa2DGetStrWidth**, **tsa2DGetFontInfo**, **tsa2DLoadFont**, and **tsa2DUnLoadFont**.
3. Use color conversion APIs: **tsa2DRGBtoYUV**, **tsa2DYUVtoRGB**, **tsa2DLoadIndexColorLUT**, **tsa2DUnLoadIndexColorLUT**, and **tsa2DGetColorFmIndex**.
4. Use drawing APIs: **tsa2DLine** (NC), **tsa2Dpoint** (NC), **tsa2DFillRect** (NC), **tsa2Dimage** (NC) to draw to the YUV422 buffer or overlay buffer or DTVCM buffer.
5. Call **tsa2Dclose** to finish.

How to Load Fonts

There are two configurations that the user can load font in: PC host and stand-alone.

PC Host

In PC host configuration, the user calls tsa2DLoadFont to load font files. In TMFont type, they are file.bit and file.tm.

stand-alone

In stand-alone configuration, you need only do the following:

```
#include "plain16.h"
pFont->fontID = &plain16;
```


There is no need to load the font.

Technical Difficulties with 2D Graphics Library

1. In the YUV422 image, two Y pixels share one set of U and V. It is difficult to render exactly two colors for two neighboring pixels and have two sharp colors next to each other.
2. For the DTVCM buffer, it uses the two least significant bits (LSBs) of U and V to indicate the blending level of video and graphics. This results in loss of colors.

Returned Error Messages

The following error messages are returned for the corresponding API.

Error code	API
TWOD_ERR_COLOR_TYPE	All the drawing APIs
TWOD_ERR_INDCOLOR_ALLOC	tsa2DLoadIndexColorLUT
TWOD_ERR_TMFONT_ALLOC	tsa2DLoadFont, tsa2DGetFontInfo
TWOD_ERR_TMFONT_MTR_FILE	tsa2DLoadFont, tsa2DGetFontInfo
TWOD_ERR_TMFONT_BIT_FILE	tsa2DLoadFont
TWOD_ERR_TMFONT_GETSTRWIDTH	tsa2DGetStrWidth
TWOD_ERR_TMFONT_NULL	tsa2DTextNC, tsa2DPolyText, tsa2DText
TWOD_ERR_TMFONT2_ALLOC	tsa2DLoadFont, tsa2DGetFontInfo
TWOD_ERR_TMFONT2_TM_FILE	tsa2DLoadFont, tsa2DGetFontInfo
TWOD_ERR_TMFONT2_BIT_FILE	tsa2DLoadFont
TWOD_ERR_TMFONT2_GETSTRWIDTH	tsa2DGetStrWidth
TWOD_ERR_TMFONT2_NULL	tsa2DTextNC, tsa2DPolyText, tsa2DText
TWOD_ERR_ALLOC	tsa2DOpen, tsa2DFillPoly
TWOD_ERR_NOT_SUPPORTED	tsa2DTextNC, tsa2DPolyText, tsa2DText, tsa2DImageNC, tsa2DPolyImage, tsa2DImage
TWOD_ERR_INVALID_RECT	tsa2DTextNC, tsa2DPolyText, tsa2DText, tsa2DFillRectNC, tsa2DPolyFillRect, tsa2DFillRect
TWOD_ERR_INVALID_POINTER	All the drawing APIs
TWOD_ERR_INVALID_FLAG	tsa2DGetFontInfo
TWOD_ERR_ODD_STRIDE	none

Error code	API
TWOD_ERR_INVALID_POLYGON	tsa2DFillPoly
TWOD_ERR_BLIT_INVALID_OPS_STRING	none
TWOD_ERR_FORMAT_MISMATCH	tsa2DBlt, tsa2DPolyBlt, tsa2DBltRegion
TWOD_ERR_TMFONT_FILENAME_LEN	tsa2DLoadFont
TWOD_ERR_TMFONT2_FILENAME_LEN	tsa2DLoadFont

2D API Data Structures

This section presents the 2D graphics API data structures. These data structures are defined in the tsa2D.h header file

Name	Page
tsa2DCapabilities_t	26
tsaYUVAColor_t	26
tsaYUVColor_t	27
tsaRGBColor_t	27
tsa2DColorType_t	28
tsa2DColor_t	28
tsa2DIndexColorLUT_t	30
tsa2DCoordinate_t	30
tsa2DRect_t	31
tsa2DImageType_t	31
tsa2DImage_t	32
tsa2DTextStyle_t	33
tsa2DFontInfoFlag_t	33
tsaFontTMCharMetrics_t	34
tsaFontTM_t	35
tsaTMFont2CharMetrics	36
tsaTMFont2	37

Name	Page
tsa2DFontType_t	38
tsa2DFont_t	38
tsa2DContext_t	39

tsa2DCapabilities_t

```
typedef struct tsa2DCapabilities_t {
   ptsaDefaultCapabilities_t   defaultCapabilities;
   tmVideoRGBYUVFormat_t       supportedBufferFormats;
} tsa2DCapabilities_t; *ptsa2DCapabilities_t;
```

Fields

`defaultCapabilities`	Default capabilities.

Description

The structure holds a list of capabilities. The 2D maintains a structure of this type to describe itself. The user can retrieve the address of this structure by calling **tsa2DGetCapabilities**.

tsaYUVAColor_t

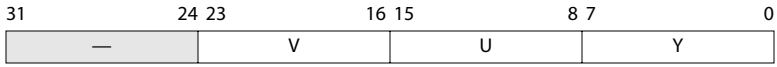
```
typedef struct tsaYUVAColor_t {
   UInt8   Y;
   UInt8   U;
   UInt8   V;
   UInt8   reserved;
} tsaYUVAColor_t, *ptsaYUVAColor_t;
```

Fields

`Y`	Y value.
`U`	U value.
`V`	V value.
`reserved`	Reserved.

Description

For the 2D display color value represent YUV values, each value takes up 8 bits of the integer value in the following order:



tsaYUVColor_t

```
typedef struct tsaYUVColor_t {
   UInt8   V;
   UInt8   U;
   UInt8   Y;
} tsaYUVColor_t, *ptsaYUVColor_t;
```

Fields

V	V value.
U	U value.
Y	Y value.

Description

For the 2D display color value represent YUV values.

tsaRGBColor_t

```
typedef struct {
   UInt8   B;
   UInt8   G;
   UInt8   R;
} tsaRGBColor_t, *ptsaRGBColor_t;
```

Fields

B	Blue color level.
G	Green color level.
R	Red color level.

Description

This structure describes RGB color.

tsa2DColorType_t

```
typedef enum {
   noColor        =  0,
   indexColor     =  1,
   YUVColor       =  2,
   YUVAColor      =  4,
   RGB888Color    =  8
   RGB565Color    = 16
   RGB555AColor   = 32
   YUVA4Color     = 64
} tsa2DColorType_t;
```

Description

This enum describes the available color type. According to **colorType** specified, **pColorData** points to particular color data. If it is **indexColor**, **pColorData** specifies the index color (i.e. an index number) of the current loaded and active index color LUT.

tsa2DColor_t

```
typedef struct {
   tsa2DColorType_t   ColorType;
   Pointer            pColorData;
} tsa2DColor_t, *ptsa2DColor_t;
```

Fields

`ColorType`	Color specified.
`pColorData`	Pointer to particular color data.

Description

According to the **ColorType** specified, **pColorData** points to particular color data. If it is **indexColor**, **pColorData** specifies the index color (for example, an index number) of the current loaded and active index color LUT.

tsaYUVA4Color_t

```
typedef struct {
   UInt8    Y;
   UInt8    U;
   UInt8    V;
   UInt8    A;
} tsaYUVA4Color_t, *ptsaYUVA4Color_t;
```

Fields

Y	Y value.
U	U value.
V	V value.
A	Alpha value: only the 4 least-significant bits are used.

Description

This structure describes a YUV color with 4-bit alpha value.

tsa2DIndexColorLUT_t

```
typedef struct tsa2DIndexColorLUT_t {
   Int32              numEntry;
   tsa2DColorType_t   LUTColorType;
   Pointer            pLUTColorData;
   UInt32             indexColorLUTID;
} tsa2DIndexColorLUT_t, *ptsa2DIndexColorLUT_t;
```

Fields

`numEntry`	Entry number.
`LUTColorType`	Color specified.
`pLUTColorData`	Color specified.
`indexColorLUTID`	Pointer to particular color data.

Description

This is the data structure used in loading the index color LUT. **numEntry** specifies the number of index colors in this LUT. **LUTColorType** specifies the color type in the look up table. **pLUTColorData** is a pointer, points to the corresponding colors in the look up table. Library fills in the **indexColorLUTID** after loading it successfully.

tsa2DCoordinate_t

```
typedef struct tsa2DCoordinate_t {
   Int   X;
   Int   Y;
} tsa2DCoordinate_t, *ptsa2DCoordinate_t;
```

Fields

`X`	X coordinate.
`Y`	Y coordinate.

Description

X and Y represent the cartesian coordinates in a 2D plane.

tsa2DRect_t

```
typedef struct tsa2DRect_t {
   tsa2DCoordinate_t   upLt;
   tsa2DCoordinate_t   btRt;
} tsa2DRect_t, *ptsa2DRect_t;
```

Fields

`upLt`	Specifies the (x,y) coordinates of the upper left position of the rectangle.
`btRt`	Specifies the (x,y) coordinates of the bottom right position of the rectangle.

Description

This data structure describes a rectangle through the positions of the upper left and bottom right coordinates.

tsa2DImageType_t

```
typedef enum {
   noImage              =  0,
   YUV422Image          =  1,
   YUV420Image          =  2,
   OverlayImage         =  4,
   BMP8BPPCLUTImage     =  8,
   PPMImage             = 16,
   GIFImage             = 32,
   RGB888Image          = 33,
   RGB565Image          = 34,
   RGB555AImage         = 35,
   YUV422A4Image        = 36,
   YUV422ChromaKeyImage = 37
} tsa2DImageType_t;
```

Description

This type definition enumerates the available image types. Only **YUV422Imagetype** is currently being supported.

tsa2DImage_t

```
typedef struct tsa2DImage_t {
   tsa2DImageType_t  imageType;
   Int               iWidth;
   Int               iHeight;
   Int               iStride;
   Pointer           pHeader;
   Pointer           pData1;
   Pointer           pData2;
   Pointer           pData3;
   Pointer           pData4;
   ptsa2DColor_t     chromaKey;
} tsa2DImage_t, *ptsa2DImage_t;
```

Fields

`imageType`	Specifies the image type.
`iWidth`	Specifies the width of the image.
`iHeight`	Specifies the height of the image.
`iStride`	Specifies the stride of the image.
`pHeader`	Pointer to the header information of the image.
`pData1`	Pointer to the first data of the image.
`pData2`	Pointer to the second data of the image.
`pData3`	Pointer to the third data of the image.
`pData4`	Pointer to the fourth data of the image.
`chromaKey`	Pointer to the color the drawing operation is to treat as transparent.

Description

This data structure provides information regarding various images. First, the user specifies image type. **pHeader** points to image header information. **pData1**, **pData2**, **pData3**, and **pData4** can be used flexibly, pointing to image data.

tsa2DTextStyle_t

```
typedef enum {
   noTextStyle     = 0,
   textOnly        = 1,
   textBackColor   = 2,
   textUnderline   = 4
} tsa2DTextStyle_t;
```

Description

This type definition enumerates the supported text styles.

Text style can be either of the following:

- **textOnly.** Draw text with foreground color.
- **textBackColor.** Draw text with foreground color and fill the background with the background color.
- **textUnderline.** Draw the text and underline with foreground color.

tsa2DFontInfoFlag_t

```
typedef enum {
   NOFONTINFOFLAG   =  0,
   MINCHAR          =  1,
   MAXCHAR          =  2,
   MAXHEIGHT        =  4,
   MAXASCENT        =  8,
   MAXDESCENT       = 16,
   MAXWIDTH         = 32,
   DEFCHAR          = 64
} tsa2DFontInfoFlag_t;
```

Description

This type definition enumerates the supported flags to get specific information regarding font, and is used in **tsa2DGetFontInfo**.

tsaFontTMCharMetrics_t

```
typedef struct tsaFontTMCharMetrics_t {
   UInt8    charNo;
   UInt8    chWidth;
   UInt8    pixWidth;
   UInt8    pixHeight;
   char     Ascent;
   char     Descent;
   UInt32   Offset;
} tsaFontTMCharMetrics_t, *ptsaFontTMCharMetrics_t;
```

Fields

`charNo`	Number of characters.
`chWidth`	Character width.
`pixWidth`	Pixel width.
`pixHeight`	Pixel height.
`Ascent`	Ascent.
`Descent`	Descent.
`Offset`	Offset to the corresponding bitmap in the bit file.

Description

TriMedia Font Character Metrics Specification of **tsaFontTM_t**.

tsaFontTM_t

```
typedef struct tsaFontTM_t {
   UInt8                       minChar;
   UInt8                       maxChar;
   UInt8                       fontType;
   UInt8                       maxHeight;
   UInt8                       maxAscent;
   UInt8                       maxDescent;
   tsaFontTMCharMetrics_t   **charMetrics;
   UInt8                     *bitmaps;
} tsaFontTM_t, *ptsaFontTM_t;
```

Fields

`minChar`	Minimum number of characters in this font set.
`maxChar`	Maximum number of characters in this font.
`fontType`	Font type.
`maxHeight`	Maximum height.
`maxAscent`	Maximum ascent.
`maxDescent`	Maximum descent.
`charMetrics`	Pointer to pointer of character metrics array.
`bitmaps`	Pointer to bitmap array.

Description

TriMedia font general data structure.

tsaTMFont2CharMetrics

```
typedef struct tsaFontTM_t {
   UInt8   charNo;
   UInt8   pixWidth;
   UInt8   firstLine;
   UInt8   lastLine;
   Int8    abcA;
   UInt8   abcB;
   Int8    abcC;
   UInt8   spare;
   UInt    offset;
} tsaTMFont2CharMetrics_t, *ptsaTMFont2CharMetrics_t;
```

Fields

`charNo`	Number of characters.
`pixWidth`	Pixel width.
`firstLine`	First line.
`lastLine`	Last line.
`abcA`	Point A.
`abcB`	Point B.
`abcC`	Point C.
`spare`	Spare.
`offset`	Offset.

Description

TriMedia font character metrics specification of **tsaTMFont2_t**.

tsaTMFont2

```
typedef struct tsaFontTM_t {
   UInt8                       fontType;
   UInt8                       minChar;
   UInt8                       maxChar;
   UInt8                       defChar;
   UInt8                       maxHeight;
   UInt8                       maxAscent;
   UInt8                       maxDescent;
   UInt8                       maxWidth;
   tsaTMFont2CharMetrics_t  **charMetrics;
   UInt8                     *bitmaps;
} tsaTMFont2_t, *ptsaTMFont2_t;
```

Fields

`fontType`	Font type.
`minChar`	Minimum number of characters in this font set.
`maxChar`	Maximum number of characters in this font.
`defChar`	Character definition.
`maxHeight`	Maximum height.
`maxAscent`	Maximum ascent.
`maxDescent`	Maximum descent.
`maxWidth`	Maximum width.
`charMetrics`	Pointer to pointer of character metrics array.
`bitmaps`	Pointer to bitmap array.

Description

TriMedia Font general data structure.

tsa2DFontType_t

```
typedef enum {
   NoFont    = 0,
   TMFont    = 1,
   TMFont2   = 2
} tsa2DFontType_t;
```

Description

This type definition enumerates the font types. Only TMFont is currently supported.

tsa2DFont_t

```
typedef struct tsa2DFont {
   tsa2DFontType_t   fontType;
   UInt32            fontID;
   Pointer           pFontPath;
} tsa2DFont_t, *ptsa2DFont_t;
```

Fields

`fontType`	Font type. Must be a member of the **tsa2DFontType_t** enum.
`fontID`	ID used internally by the library. You shouldn't set this field.
`pFontPath`	Base name of font, including relative or absolute path (e.g., **"../../data/fonts/plain20"**).

Description

The user specifies **fontType** and **fpontPath** to locate the font file. Once this font is loaded, the library fills in the fontID. Only TMFont **fontType** is currently supported.

tsa2DContext_t

```
typedef struct tsa2DContext {
   ptsa2DColor_t   pPointColor;
   ptsa2DColor_t   pLineColor;
   ptsa2DColor_t   pFillColor;
   ptsa2DColor_t   pTextColor;
   ptsa2DColor_t   pBgColor;
   UInt32          lineStyle;
   UInt32          textStyle;
   UInt32          fillStyle;
   UInt32          bltStyle;
} tsa2DContext_t, *ptsa2DContext_t;
```

Fields

`pPointColor`	Color used in drawing the point.
`pLineColor`	Color used in drawing the line.
`pFillColor`	Color used in drawing the fill the rectangle.
`pTextColor`	Color used in drawing the text.
`pBgColor`	Color used in drawing the background.
`lineStyle`	Line style.
`textStyle`	Text style.
`fillStyle`	Fill rectangle style.
`bltStyle`	Bullet style.

Description

This graphic context data structure contains graphic context information that is used in various APIs.

2D API Functions

This section presents the 2D API data functions. These data functions are defined in the tsa2D.h header file.

Name	Page
tsa2DGetCapabilities	42
tsa2DOpen	43
tsa2DClose	43
tsa2DRGBtoYUV	44
tsa2DYUVtoRGB	45
tsa2DLoadIndexColorLUT	46
tsa2DUnLoadIndexColorLUT	47
tsa2DGetColorFmIndex	48
tsa2DPointNC	49
tsa2DLineNC	50
tsa2DFillRectNC	51
tsa2DImageNC	52
tsa2DTextNC	53
tsa2DSetPixel	55
tsa2DGetPixel	56
tsa2DPoint	57
tsa2DLine	58
tsa2DFillRect	59
tsa2DFillPoly	60
tsa2DImage	61
tsa2DText	62
tsa2DBlt	64
tsa2DBltRegion	65
tsa2DPolyPoint	66
tsa2DPolyLine	67

Name	Page
tsa2DPolyFillRect	68
tsa2DPolyImage	69
tsa2DPolyText	70
tsa2DPolyBlt	72
tsa2DGetStrWidth	73
tsa2DGetFontInfo	74
tsa2DTMFontSetCharSpacingInString	75
tsa2DTMFontGetCharSpacingInString	76
tsa2DLoadFont	77
tsa2DUnLoadFont	78

tsa2DGetCapabilities

```
tmLibappErr_t tsa2DGetCapabilities(
   ptsa2DCapabilities_t  *pCap
);
```

Parameters

pCap	Pointer to variable in which to return a pointer to capabilities data.

Return Codes

TMLIBDEV_OK	Success.

Description

Retrieves global and 2D capabilities.

tsa2DOpen

```
tmLibappErr_t tsa2DOpen(
   Int   *instance
);
```

Parameters

`instance`	Pointer to the (returned) instance.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_ALLOC`	The function failed to allocate memory.

Description

User calls **tsa2DOpen** to get an instance ID. This function assigns a unique 2D instance to the caller.

tsa2DClose

```
tmLibappErr_t tsa2DClose(
   Int   instance
);
```

Parameters

`instance`	The instance to close.

Return Codes

`TMLIBDEV_OK`	Success.

Description

User calls **tsa2DClose** when exit. This routine deallocates the 2D instance.

tsa2DRGBtoYUV

```
tmLibappErr_t tsa2DRGBtoYUV(
   Int     instance,
   UInt8   r,
   UInt8   g,
   UInt8   b,
   UInt8   *y,
   UInt8   *u,
   UInt8   *v
);
```

Parameters

`instance`	Instance.
`r`	Red value.
`g`	Green value.
`b`	Blue value.
`y`	Y value.
`u`	U value.
`v`	V value.

Return Codes

`TMLIBDEV_OK`	Success.

Description

The function takes in RGB color and converts it to YUV. The returned Y, U, V, values are placed in ***y**, ***u**, and ***v**, respectively. The values are restricted to the range 16–35.

tsa2DYUVtoRGB

```
tmLibappErr_t tsa2DYUVtoRGB(
   Int     instance,
   UInt8   y,
   UInt8   u,
   UInt8   v,
   UInt8   *r,
   UInt8   *g,
   UInt8   *b
);
```

Parameters

`instance`	Instance.
`y`	Y value.
`u`	U value.
`v`	V value.
`r`	Red value.
`g`	Green value.
`b`	Blue value.

Return Codes

`TMLIBDEV_OK`	Success.

Description

The function takes in YUV values and converts them to RGB. The returned R, G, B values are placed in ***r**, ***g**, and ***b**, respectively.

tsa2DLoadIndexColorLUT

```
tmLibappErr_t tsa2DLoadIndexColorLUT(
   Int                     instance,
   ptsa2DIndexColorLUT_t   pIndClr
);
```

Parameters

`instance`	Instance.
`pIndClr`	Pointer to the index color LUT. The user specifies: 1. Number of entries (**numEntry**) in the index color. 2. The corresponding LUT color type (**LUTColor-Type**). 3. Pointer to the corresponding array of colors (**pLUTColorData**). Library returns **indexColorLUTID**.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_INDCOLOR_ALLLOC`	The function failed in memory allocation.

Description

This routine loads user's index color Look Up Table (LUT) to the 2D Library.

tsa2DUnLoadIndexColorLUT

```
tmLibappErr_t tsa2DUnLoadIndexColorLUT(
   Int                     instance,
   ptsa2DIndexColorLUT_t   pIndClr
);
```

Parameters

`instance`	The instance.
`pIndClr`	Pointer to the index color LUT. Library unloads this index color LUT in the library.

Return Codes

`TMLIBDEV_OK`	Success.

Description

This routine unloads the specified index color Look Up Table (LUT) in the 2D Library.

tsa2DGetColorFmIndex

```
tmLibappErr_t tsa2DGetColorFmIndex(
   Int                     instance,
   Int                     index,
   ptsa2DIndexColorLUT_t   pIndexCLUT,
   ptsa2DColor_t           pColor
);
```

Parameters

`instance`	The instance.
`index`	Index in the index color look up table (LUT).
`pIndexCLUT`	Pointer to the index color LUT.
`pColor`	Pointer to **ptsa2DColor_t**.

Return Codes

`TMLIBDEV_OK`	Success.

Description

This function returns the color in **pColor** according to the specified index number in the index and index color look up table in the pIndexCLUT.

The user sets the index color number, specifies the index color look up table to be used, allocates space on pColor. The function gets the corresponding color from the CLUT and put those color values in **pColor**. Only YUV color type is currently supported.

tsa2DPointNC

```
tmLibappErr_t tsa2DPointNC(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPoint,
   ptsa2DColor_t        pColor
);
```

Parameters

instance	The instance.
pPacket	Pointer to input buffer packet header tmAvFormats.h and packet data.
pPoint	Pointer to coordinate of a point within the input buffer.
pColor	Color to draw the point.

Return Codes

TMLIBDEV_OK	Success.
TWOD_ERR_OUT_OF_BOUNDARY	The point specified is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function draws a point in the input packet buffer with specified coordinate and color.

tsa2DLineNC

```
tmLibappErr_t tsa2DLineNC(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DCoordinate_t   pPt2,
   ptsa2DVOColor_t      pColor
);
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to input buffer packet header and packet data.
`pPt1`	Pointer to point 1.
`pPt2`	Pointer to point 2.
`pColor`	The color of the line.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_UPLT_BTRT`	Error in the upper left and bottom right coordinates specification.
`TWOD_ERR_OUT_OF_BOUNDARY`	The line specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function draws a line from point 1 to point 2, with color specified by **pColor**, into the input packet buffer.

tsa2DFillRectNC

```
tmLibappErr_t tsa2DFillRectNC(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DCoordinate_t   pPt2,
   ptsa2DVOColor_t      pColor
);
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to input buffer packet header and packet data.
`pPt1`	Pointer to upper left point.
`pPt2`	Pointer to bottom right point.
`pColor`	The color of the rectangle.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_INVALID_RECT`	The rectangle specified is invalid.
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function fills a rectangle in the input packet buffer according to the rectangle specification of the upper left and the bottom right coordinates, and the rectangle's fill color.

tsa2DImageNC

```
tmLibappErr_t tsa2DImageNC(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DCoordinate_t   pPt2,
   ptsa2DImage_t        pImage
);
```

Parameters

instance	The instance.
pPacket	Pointer to input buffer packet header and packet data.
pPt1	Pointer to top left point.
pPt2	Pointer to bottom right point.
pImage	Pointer to image.

Return Codes

TMLIBDEV_OK	Success.
TWOD_ERR_UPLT_BTRT	Error in the upper left and bottom right coordinates specification.
TWOD_ERR_OUT_OF_BOUNDARY	The rectangle specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function displays an image to the input packet buffer (**pPacket**) according to the rectangle specified in the upper left and bottom right coordinates.

tsa2DTextNC

```
tmLibappErr_t tsa2DTextNC(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt,
   const char          *string,
   ptsa2DFont_t         pFont,
   ptsa2DColor_t        pFColor,
   ptsa2DColor_t        pBColor,
   ptsa2DTextStyle_t    textStyle
);
```

Parameters

`instance`	Instance.
`pPacket`	Pointer to input buffer packet header and packet data.
`pPt`	Pointer to (x, y) of starting position.
`string`	A string of characters to be drawn; can be one single character.
`pFont`	Pointer to font structure being used.
`pFColor`	Character string is drawn with this foreground color.
`pBColor`	Background is filled with this color. User should supply a valid background color, even if it is not used.
`textStyle`	Character string is drawn with this text style. See tsa2DTextStyle_t. 1. **textOnly**—draw text with foreground color. 2. **textBackColor**—draw text with the foreground color and fill the background with the background color. 3. **textUnderline**—draw the text and underline with the foreground color.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_TMFONT_NULL`	The TMFont pointer is null.
`TWOD_ERR_TMFONT2_NULL`	The TMFont2 pointer is null.
`TWOD_ERR_NOT_SUPPORTED`	The specified font type or text style are not supported.
`TWOD_ERR_INVALID_RECT`	Error in the rectangle coordinates specification.

TWOD_ERR_OUT_OF_BOUNDARY	Returned if the rectangle specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function draws a string of characters in the input buffer **pPacket** by specifying the (x,y) coordinate **pPt**. The specified starting position is the base point (point between ascent and descent of a character) of the first character in the string. It supports two font types (**TMFont** and **TMFont2**), and three text drawing styles (**textOnly**, **textBackColor**, **textUnderline**). User also specifies the desired background and foreground color.

tsa2DSetPixel

```
tmLibappErr_t tsa2DSetPixel(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPixelSet,
   ptsa2DContext_t      pContext
);
```

Parameters

`instance`	Instance.
`pPacket`	Pointer to the input buffer packet header and packet data.
`pPixelSet`	Pointer to coordinate of a pixel.
`pContext`	Pointer to 2D context. **pPointColor** of **pContext** is the color to be used to set the pixel color.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_OUT_OF_BOUNDARY`	The point specified is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function sets a pixel in the packet buffer with the **pPointColor** of the **pContext**.

tsa2DGetPixel

```
tmLibappErr_t tsa2DGetPixel(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPixelGet,
   ptsa2DContext_t      pContext
);
```

Parameters

`instance`	Instance.
`pPacket`	Pointer to the input buffer packet header and packet data.
`pPixelGet`	Pointer to coordinate of a pixel.
`pContext`	Return color in the pPointColor of pContext when success.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_OUT_OF_BOUNDARY`	The point specified is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function gets a pixel color of a specified position in the packet buffer.

tsa2DPoint

```
tmLibappErr_t tsa2DPoint(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DContext_t      pContext
);
```

Parameters

`instance`	Instance.
`pPacket`	Pointer to buffer information, type, and data.
`pPt1`	Pointer to 2D point.
`pContext`	Pointer to 2D context.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_OUT_OF_BOUNDARY`	The point specified is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function draws a point to the specified position in the packet buffer with the **pPoint-Color** of the **pContext**.

tsa2DLine

```
tmLibappErr_t tsa2DLine(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DCoordinate_t   pPt2,
   ptsa2DContext_t      pContext
);
```

Parameters

instance	Instance.
pPacket	Pointer to buffer information, type, and data.
pPt1	Pointer to first 2D point.
pPt2	Pointer to end 2D point.
pContext	Pointer to 2D context line color.

Return Codes

TMLIBDEV_OK	Success.
TWOD_ERR_UPLT_BTRT	Error in the upper left and bottom right coordinates specification.
TWOD_ERR_OUT_OF_BOUNDARY	The line specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function draws a line in the input packet buffer with the **pLinrColor** of the **pContext**.

tsa2DFillRect

```
tmLibappErr_t tsa2DFillRect(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DCoordinate_t   pPt2,
   ptsa2DContext_t      pContext
);
```

Parameters

instance	Instance.
pPacket	Pointer to buffer info, type and data.
pPt1	Pointer to top left point.
pPt2	Pointer to bottom right point.
pContext	Pointer to 2D context.

Return Codes

TMLIBDEV_OK	Success.
TWOD_ERR_INVALID_RECT	The rectangle specified is invalid.
TWOD_ERR_OUT_OF_BOUNDARY	The rectangle specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function fills a rectangle in the input packet buffer according to the rectangle specification, and the **pFillColor** of the **pContext**.

tsa2DFillPoly

```
tmLibappErr_t tsa2DFillPoly(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPolyPoints,
   Int                  numPoints,
   ptsa2DContext_t      pContext
);
```

Parameters

`instance`	Instance.
`pPacket`	Pointer to buffer info, type and data.
`pPolyPoints`	Pointer to a list of points that form a polygon.
`numPoints`	Number of points in the polygon.
`pContext`	Pointer to 2D context.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_ALLOC`	The function failed in memory allocation.
`TWOD_ERR_INVALID_POLYGON`	The polygon specified is invalid.
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function fills a convex polygon in the input packet buffer according to the polygon specification, and the **pFillColor** of the **pContext**.

tsa2DImage

```
tmLibappErr_t tsa2DImage(
   Int                  instance,
   ptmAvPacket_t        pPacket,
   ptsa2DCoordinate_t   pPt1,
   ptsa2DCoordinate_t   pPt2,
   ptsa2DImage_t        pImage,
   ptsa2DContext_t      pContext
);
```

Parameters

`instance`	Instance.
`pPacket`	Pointer to buffer info, type, and data.
`pPt1`	Pointer to top left point.
`pPt2`	Pointer to bottom right point.
`pImage`	Pointer to image.
`pContext`	Pointer to 2D context.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_UPLT_BTRT`	Error in the upper left and bottom right coordinates specification.
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function copies an image to the input packet buffer (**pPacket**) according to the rectangle specified in the upper left and bottom right coordinates.

tsa2DText

```
tmLibappErr_t tsa2DText(
   Int                  instance,
   ptmAvPacket_         pPacket,
   ptsa2DCoordinate_t   pPt,
   const char          *str,
   ptsa2DFont_t         pFont,
   ptsa2DContext_t      pContext
);
```

Parameters

instance	Instance.
pPacket	Pointer to input buffer packet header and packet data.
pPt	Pointer to (x, y) of starting position.
*str	Pointer to a string of characters to be drawn; can be one single character.
pFont	Pointer to a valid font.
pContext	Pointer to the 2D Context text and background color.

Return Codes

TMLIBDEV_OK	Success.
TWOD_ERR_TMFONT_NULL	The TMFont pointer is null.
TWOD_ERR_TMFONT2_NULL	The TMFont2 pointer is null.
TWOD_ERR_NOT_SUPPORTED	The specified font type or text style are not supported.
TWOD_ERR_INVALID_RECT	Error in the rectangle coordinates specification.
TWOD_ERR_OUT_OF_BOUNDARY	The rectangle specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function draws a string of characters in the input buffer **pPacket** by specifying the (x,y) coordinate **pPt**. The specified starting position is the base point (point between ascent and descent of a character) of the first charcater in the string. It supports two font types (TMFont and TMFont2), and three text drawing styles (**textOnly**, **textBackColor**,

textUnderline). It uses the **pTextColor** of the **pContext** as the text foreground color and it uses the **pBgColor** of the **pContext** as the background color.

tsa2DBlt

```
tmLibappErr_t tsa2DBlt(
   Int               instance,
   ptmAvPacket_t     pDstPacket,
   ptmAvPacket_t     pSrcPacket,
   ptmAvPacket_t     pDstStartPt,
   ptsa2DContext_t   pContext
);
```

Parameters

instance	Instance.
pDstPacket	Pointer to destination buffer.
pSrcPacket	Pointer to source buffer.
pDstStartPt	Pointer to start (x, y) in destination buffer.
pContext	Pointer to context information.

Return Codes

TMLIBDEV_OK	Success.
TWOD_ERR_NOT_SUPPORTED	The specified font type or text style are not supported .
TWOD_ERR_OUT_OF_BOUNDARY	The rectangle specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function copies the entire source buffer **pSrcPacket** to the specified location in the destination packet buffer **pDstPacket**.

tsa2DBltRegion

```
tmLibappErr_t tsa2DBlt(
   Int               instance,
   ptmAvPacket_t     pDstPacket,
   ptmAvPacket_t     pSrcPacket,
   ptsaCoordinate_t  pDstStartPt,
   ptsaCoordinate_t  pSrcStartPt,
   Int               width,
   Int               height,
   ptsa2DContext_t   pContext
   Int               ops
);
```

Parameters

`instance`	Instance.
`pDstPacket`	Pointer to destination buffer.
`pSrcPacket`	Pointer to source buffer.
`pDstStartPt`	Pointer to start (x, y) in destination buffer.
`pSrcStartPt`	Pointer to start (x, y) in source buffer.
`width`	Width of the region to be BLT'd.
`height`	Height of the region to be BLT'd.
`pContext`	Pointer to context information. Only the YUV422 buffer type is supported and pContext is not currently used.
`ops`	Logical operation to be performed on the source and destination pixels.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_NOT_SUPPORTED`	The specified font type or text style are not supported.
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function copies the source buffer with the specified starting position and (width, height) to the destination packet buffer at the sepcified destination starting position.

tsa2DPolyPoint

```
tmLibappErr_t tsa2DPolyPoint(
   Int                  instance,
   ptmAvPacket_        *pPktList,
   Int                  numPkt,
   ptsa2DCoordinate_t   pPtList,
   Int                 *pNumPerPk,
   ptsa2DColor_t        pColor
);
```

Parameters

`instance`	Instance.
`*pPktList`	Pointer to an array of packet pointers.
`numPkt`	Number of packets to pass in.
`pPtList`	Pointer to an array of 2D points.
`pNumPerPk`	Pointer to array of **Int** which specifies the number of points to be drawn in each packet.
`pColor`	Color pointer, the **pColor->pColorData** is a pointer to an array of 2D colors.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_OUT_OF_BOUNDARY`	The point specified is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function draws multiple numbers of points on multiple numbers of packets according to the supplied positions and colors.

tsa2DPolyLine

```
tmLibappErr_t tsa2DPolyLine(
   Int                  instance,
   ptmAvPacket_t       *pPktList,
   Int                  numPkt,
   ptsa2DCoordinate_t   pPt1List,
   ptsa2DCoordinate_t   pPt2List,
   Int                  pNumPerPkt,
   ptsa2DColor_t        pColor
);
```

Parameters

`instance`	Instance.
`pPktList`	Pointer to an 'array' of packet pointers.
`numPkt`	Number of packets to pass in.
`pPt1List`	Pointer to an array of beginning 2D points.
`pPt2List`	Pointer to an array of ending 2D points.
`pNumPerPkt`	Pointer to array of `Int` that specifies number of lines to be drawn in each packet.
`pColor`	Color pointer, the **pColor->pColorData** is a pointer to an array of 2D colors.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_UPLT_BTRT`	Error in the upper left and buttom right coordinates specification.
`TWOD_ERR_OUT_OF_BOUNDARY`	The line specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function draws multiple numbers of lines on multiple numbers of packets according to the supplied lines and colors.

tsa2DPolyFillRect

```
tmLibappErr_t tsa2DPolyFillRect(
   Int                  instance,
   ptmAvPacket_t       *pPktList,
   Int                  numPkt,
   ptsa2DCoordinate_t   pPt1List,
   ptsa2DCoordinate_t   pPt2List,
   Int                 *pNumPerPkt,
   ptsa2DColor_t    pColor
);
```

Parameters

`instance`	Instance.
`pPktList`	Pointer to an array of packet pointers.
`numPkt`	Number of packets to pass in.
`pPt1List`	Pointer to an array of upper left 2D points.
`pPt2List`	Pointer to an array of bottom right 2D points.
`pNumPerPkt`	Pointer to array of **Int** which specifies number of fill-rectangles to be drawn in each packet.
`pColor`	Color pointer, the **pColor->pColorData** is a pointer to an array of 2D colors.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_INVALID_RECT`	The rectangle specified is invalid.
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function fills multiple numbers of rectangles on multiple numbers of packets according to the supplied rectangles and colors.

tsa2DPolyImage

```
tmLibappErr_t tsa2DPolyImage(
   Int                  instance,
   ptmAvPacket_t       *pPktList
   Int                  numPkt
   ptsa2DCoordinate_t   pPt1List
   ptsa2DCoordinate_t   pPt2List
   Int                 *pNumPerPkt
   ptsa2DImage_t   *pImageList
);
```

Parameters

`instance`	Instance.
`pPktList`	Pointer to an array of packet pointers.
`numPkt`	Number of packets to pass in.
`pPt1List`	Pointer to an array of beginning 2D points..
`pPt2List`	Pointer to an array of ending 2D points.
`pNumPerPkt`	Pointer to array of **Int** which specifies number of lines to be drawn in each packet.
`pImageList`	Pointer to an array of image pointers.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_UPLT_BTRT`	Error in the upper left and bottom right coordinates specification.
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function copies multiple numbers of images to multiple numbers of packets according to the supplied images and destination rectangle locations.

tsa2DPolyText

```
tmLibappErr_t tsa2DPolyText(
   Int                  instance,
   ptmAvPacket_t       *pPktList,
   ptsa2DCoordinate_t   pPtList,
   const char         **string,
   ptsa2DFont_t        *pFontList,
   ptsa2DColor_t        pFColor,
   ptsa2DColor_t        pBColor,
   tsa2DTextStyle_t    *textStyle,
   Int                 *pNumPerPkt
);
```

Parameters

`instance`	Instance.
`pPktList`	Pointer to an array of packet pointers.
`pPktList`	Number of packets to pass in.
`pPtList`	Pointer to an array of starting positions.
`string`	Pointer to an array of string of characters to be drawn.
`pFontList`	Pointer to an array of loaded fonts.
`pFColor`	foreground (or text) color pointer, the **pColor->pColorData** is a pointer to an array of 2D colors.
`pBColor`	Background is filled with this color. User should supply a valid background color, even if it is not used.
`textStyle`	Pointer to an array of text styles. It can be either: **textOnly** (drawstext with foreground color), **textBackColor** (draws text with foreground color and fill the back with background color), or **textUnderline** (draws the text and underline with foreground color).
`pNumPerPkt`	Pointer to array of **Int** which specifies number of lines to be drawn in each packet.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_TMFONT_NULL`	The TMFont pointer is null.
`TWOD_ERR_TMFONT2_NULL`	The TMFont2 pointer is null.
`TWOD_ERR_NOT_SUPPORTED`	The specified font type or text style are not supported.

TWOD_ERR_INVALID_RECT	Error in the rectangle coordinates specification.
TWOD_ERR_OUT_OF_BOUNDARY	The rectangle specification is out of the packet boundary.
TWOD_ERR_INVALID_POINTER	The function encounters an invalid pointer.
TWOD_ERR_COLOR_TYPE	The color type is not consistent with the packet buffer type.

Description

The function draws multiple strings of characters to multiple numbers of packets according to the supplied input information.

tsa2DPolyBlt

```
tmLibappErr_t tsa2DPolyBlt(
   Int                 instance,
   ptmAvPacket_t      *pDstPktList,
   ptmAvPacket_t      *pSrcPktList,
   int                 numPkt,
   ptsa2DCoordinate_t  pDstStartPtList,
   ptsa2DCoordinate_t  pSrcStartPtList,
   Int                *pNumPerPkt,
   Int                *pWidthList,
   Int                *pHeightList,
   ptsa2DContext_t   pContext
);
```

Parameters

`instance`	Instance.
`pDstPktList`	Pointer to an array of dst packet pointers.
`pSrcPktList`	Pointer to an array of src packet pointers.
`numPkt`	number of packet pass in.
`pDstStartPtList`	Pointer to an array of destination (**dst**) starting points.
`pSrcStartPtList`	Pointer pointer to an array of source (**src**) starting points.
`pNumPerPkt`	Pointer to array of int which specifies number of fill-rectangles to be drawn in each packet.
`pWidthList`	Pointer to an array of width.
`pHeightList`	Pointer to an array of height.
`pContext`	Pointer to context. This is not used currently.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_NOT_SUPPORTED`	The specified font type or text style are not supported .
`TWOD_ERR_OUT_OF_BOUNDARY`	The rectangle specification is out of the packet boundary.
`TWOD_ERR_INVALID_POINTER`	The function encounters an invalid pointer.
`TWOD_ERR_COLOR_TYPE`	The color type is not consistent with the packet buffer type.

Description

The function copies a number of rectangles from the source to the destination. User specifies the source and destination starting points and width and height for each Blt.

tsa2DGetStrWidth

```
tmLibappErr_t tsa2DGetStrWidth(
   Int           instance,
   const char   *string,
   Int          *width,
   ptsa2DFont_t  pFont
);
```

Parameters

`instance`	Instance.
`string`	String for which to get the pixel width.
`width`	The calculated pixel width, returning to caller.
`pFont`	Pointer to a valid font.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_TMFONT_GETSTRWIDTH`	The function failed on TMFont font type.
`TWOD_ERR_TMFONT2_GETSTRWIDTH`	The function failed on TMFont2 font type.

Description

The function gets the width, in pixels, of the passed string.

tsa2DGetFontInfo

```
tmLibappErr_t tsaGetFontInfo(
   Int                  instance,
   tsa2DFontInfoFlag_t  flag,
   Int                 *retVal,
   ptsa2DFont_t         pFont
);
```

Parameters

`instance`	Instance.
`flag`	Flag to indicate the requested font entry.
`retVal`	Return value to caller.
`pFont`	Pointer to a valid font.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_TMFONT_ALLOC`	Failed in TMFont alloc.
`TWOD_ERR_TMFONT2_ALLOC`	Failed in TMFont2 alloc.
`TWOD_ERR_TMFONT_MTR_FILE`	Failed in reading information from *.mtr file on TMFont type of font.
`TWOD_ERR_TMFONT2_TM_FILE`	When failed in reading information from *.tm file on TMFont2 type of font.
`TWOD_ERR_INVALID_FLAG`	When invalid flag passed in.

Description

The function gets the specific font information according to the specified flag value.

tsa2DTMFontSetCharSpacingInString

```
tmLibappErr_t tsa2DTMFontSetCharSpacingInString(
   Int   instance,
   Int   spacingTMFont,
);
```

Parameters

`instance`	Instance.
`spacingTMFont`	Value of spacing to be set on the TMFont.

Return Codes

`TMLIBDEV_OK`	Success.

Description

Sets the spacing between characters in a string. The default is 2. This applies only to the TMFont type of fonts.

tsa2DTMFontGetCharSpacingInString

```
tmLibappErr_t tsa2DTMFontGetCharSpacingInString(
   Int    instance,
   Int   *spacingTMFont,
);
```

Parameters

`instance`	Instance.
`spacingTMFont`	Value of spacing to be retrieved on the TMFont.

Return Codes

`TMLIBDEV_OK`	Success.

Description

Gets the spacing between characters in a string. The default is 2. This applies only to the TMFont type of fonts.

tsa2DLoadFont

```
tmLibappErr_t tsa2DLoadFont(
   Int            instance,
   ptsa2DFont_t   pFont
);
```

Parameters

`instance`	Instance.
`pFont`	Pointer to **tsa2DFont_t**. The user provides information regarding font type and font path. Library loads in the specified font and return a **fontID** in the **tsa2DFont_t** structure.

Return Codes

`TMLIBDEV_OK`	Success.
`TWOD_ERR_TMFONT_ALLOC`	Failed in TMFont alloc.
`TWOD_ERR_TMFONT2_ALLOC`	Failed in TMFont2 alloc.
`TWOD_ERR_TMFONT_MTR_FILE`	Failed in reading information from *.mtr file on TMFont type of font.
`TWOD_ERR_TMFONT2_TM_FILE`	Failed in reading information from *.tm file on TMFont2 type of font.
`TWOD_ERR_TMFONT_BIT_FILE`	Failed in reading information from *.bit file on TMFont type of font.
`TWOD_ERR_TMFONT2_BIT_FILE`	Failed in reading information from *.bit file on TMFont2 type of font.
`TWOD_ERR_TMFONT_FILENAME_LEN`	The expanded name of a TMFont .mtr or .bit file is too long.
`TWOD_ERR_TMFONT2_FILENAME_LEN`	The expanded name of a TMFont2 .tm or .bit file is too long.

Description

The function loads the font specified in the font path to the 2D Library.

tsa2DUnLoadFont

```
tmLibappErr_t tsaUnLoadFont(
   Int            instance,
   ptsa2DFont_t   pFont
);
```

Parameters

`instance`	Instance.
`pFont`	Pointer to **tsa2DFont_t**. Library looks up the font type on the **fontID** in the **tsa2DFont_t** struct and unloads it.

Return Codes

`TMLIBDEV_OK`	Success.

Description

The function unloads the font specified by **pFont** from the 2D Library.

Chapter 3

Closed-Captioning (EIA-608) API

Topic	Page
DTVCC Decoder (EIA-608) Overview	80
Operation	81
Sample Application	82
VrendEia608 API Functions	83
VrendEia608 API Enumerations and Data Structures	96

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

DTVCC Decoder (EIA-608) Overview

The TriMedia VrendEia608 library fulfills the requirements contained in *Recommended Practice For Line 21 Data Service (EIA-608A)*. The line 21 data, compatible to EIA608A recommendation, consists of data on field 1 and field 2 of the video signal. Either field can contain data for more than one channel.

Field 1 contains the following channels.

- CC1 (Primary Synchronous Caption Service)
- CC2 (Special Non-synchronous Use Captions)
- T1 (First Text Service)
- T2 (Second Text Service).

Field 2 contains the following channels.

- CC3 (Secondary Synchronous Caption Service)
- CC4 (Special Non-synchronous Use Captions)
- T3 (Third Text Service)
- T4 (Fourth Text Service)
- XDS (Extended Data Service)
- ATVEF service

The **VrendEia608** component decodes those channels and displays the result using the video out unit of the TriMedia chip. The library makes use of the 2D library and of the window manager, which makes it possible to overlap non-**VrendEia608** specific information like on-screen display (OSD).

The library retrieves the required data from the input pin. The **VrendEia608** library is a renderer and therefore does not provide any output pin. The decoded information is placed in a pre-allocated output buffer. The example application **exolVrendEia608** demonstrates how to use the **VrendEia608** library.

The performance of the VrendEia608 decoder can be parted in different categories. Since the processor load largely depends on the decoded data, worst case scenarios and test tape performances are published. The decoding itself does not require much effort—the performance-consuming part is the rendering. Only the changes on the screen force the display driver of the VrendEia608 component to re-render the screen. Therefore, a scrolling, flashing text in the biggest window available needs the highest processor load.

Below are some results:

EIA/CEG Test Tape for Closed Captioning proposal, version 3.0 DECODED (record Date 23 January 1992) gives an average processor load of 1.6 MHz.

worst case processor load flashing, scrolling text mode with eight displayed rows gives a processor load of 21.2 MHz. The code sequence for this worst case is:

```
14 2A 14 28 "01abcdefghijklmnopqrstuvwxyz123
14 2D 14 28 "02abcdefghijklmnopqrstuvwxyz123
.
.
14 2D 14 28 "2Eabcdefghijklmnopqrstuvwxyz123
14 2D 14 28 "2Fabcdefghijklmnopqrstuvwxyz123
```

Operation

After creating an instance with **tmolVrendEia608Open** the component is ready for setup with the required parameters by calling **tmolVrendEia608InstanceSetup**. During the opening process an instance of the 2D library and of the window manager is created.

The default values of the VrendEia608 component can be retrieved by calling **tmolVrendEia608GetInstanceSetup**. After that it is easier to call the **tmolVrendEia608InstanceSetup** function, because the setup structure tree is allocated by the library and only some instance specific parameters have to be modified. Among the default values, the instance setup function needs to get two handles to the two different fonts the EIA-608A standard requires. One plain font and one italic font. Depending on the font size the output area size will be calculated based on the character dimensions and the number of columns and rows. EIA-608A has 32 columns and 15 rows. It is recommended to use a font with a character width of 16 pels (TV-pixels) and a height of 26 scanlines. The height of the font must be always multiple of 13. The field **pBackPlane** of the instance setup function structure has to be initialized. This parameter is used by the VrendEia608 component to initialize the window manager.

After this procedure the decoder is ready to run by calling **tmolVrendEia608Start**. After that the decoding is running in a separate task. The component retrieves EIA-608A data from the input pin and renders the decoded data to the back-plane it received during the setup phase. The decoder allocates a memory saving virtual window for its operation. The input format of the packets arriving at the input pin, is a generic format. One packet consists of only one buffer. The size of this buffer is four bytes. Byte one contains the type of the retrieved data. It can be either data or others. Byte two marks the next two bytes as valid or invalid. Byte three represents character one and byte four represents character two of a line 21. The decoder handles invalid line 21 data bytes even if the byte two says that the data bytes are valid.

During decoding it is always possible to change some settings of the decoder by calling **tmolVrendEia608InstanceConfig**. The user can choose between the services EIA-608A provides. Furthermore it can be retrieved what services are currently available.

The EIA-608A standard specifies some features that are real-time related and therefore synchronized to the video output. If the application wants to update the display content in a synchronous way it calls the **tmolVrendEia608FieldSync** function. Using this function flashing and scrolling are possible in a field synchronous way.

The new data service ATVEF is supported by decoding the included ATVEF data from the Text 2 channel. That means only URL addresses that are transmitted will be decoded. As soon as a complete address arrives the VrendEia608 decoder calls the progress function (if the progress report flag was set in the instance setup function). With the URL string a time-stamp will be passed to the application. This is done only if the incoming EIA-608 data already contained valid time stamps. To use the ATVEF decoding capabilities an ATVEF buffer has to be provided using the instance setup function.

Sample Application

The example application demonstrates the available features of the EIA-608A decoder. It is possible to change some settings when the VrendEia608 component is running by issuing commands at the prompt. The following commands are available and are also briefly described by the help output.

`logo`	This command enables and disables the TriMedia DTV logo. This feature demonstrates the clipping functionality of the VrendEia608 library. If the logo is on, the entire screen is divided into four sub-windows.
`services`	This command retrieves information from the EIA-608A decoder. It displays which services are or aren't available.
`cc<number>`	This command tells the decoder which closed caption service to decode. Valid numbers are between 1 through 4. If the decoder was previously disabled ('off') it is switched on.
`t<number>`	This command tells the decoder which text service to decode. Valid number are 1 through 4. If the decoder was previously disabled ('off') it is switched on.
`off`	This command switches the decoder off.
`xds`	This command enables the extended data service functionality.
`wx<number>`	This command specifies the horizontal location of the display window. The number represents the number of pels.
`wy<number>`	This command specifies the vertical location of the display window. The number represents the number of scanlines.
`stopinput`	This commands stops the input file reader. It is used to test interruptions on the input.
`startinput`	This commands restarts the file reader. It is used to test interruptions on the input.

`stress`	This command issues a rapid sequence of 'off' and 'cc1' commands. It is used to test stability of the decoder in terms of enabling and disabling.

During start-up the following command line program arguments are checked.

`infile <filename>`	This argument specifies an input file (if not specified the default input file is used: t_jan.bin). If not specified, the default search path is ../../data/eia608.
`wy<number>`	See above.
`wx<number>`	See above.
`cc<number>`	See above.
`t<number>`	See above.

VrendEia608 API Functions

This section describes the functions contained in the VrendEia608 API.

Name	Page
tmalVrendEia608Open	84
tmolVrendEia608Open	85
tmalVrendEia608Close	86
tmolVrendEia608Close	86
tmalVrendEia608Start	87
tmolVrendEia608Start	87
tmalVrendEia608Stop	88
tmolVrendEia608Stop	88
tmalVrendEia608GetCapabilities	89
tmolVrendEia608GetCapabilities	89
tmolVrendEia608GetInstanceSetup	90
tmalVrendEia608InstanceConfig	90
tmolVrendEia608InstanceConfig	91
tmalVrendEia608InstanceSetup	92
tmolVrendEia608InstanceSetup	92
tmalVrendEia608RedrawFunc	93
tmalVrendEia608DecodePacket	94
tmalVrendEia608FieldVsync	95

tmalVrendEia608Open

```
tmLibappErr_t tmalVrendEia608Open(
   Int   *instance
)
```

Parameters

`instance`	Pointer (returned) to the instance.

Description

This function creates an instance and returns a pointer to the opened instance. The error value must be checked to decide whether the instance value is valid or not. The **Vrend-Eia608** library uses the 2D library and creates an instance of this library during Open. If an error occurred during the creation of the 2D library instance, it is returned as the return value of the Open function.

tmolVrendEia608Open

```
extern tmLibappErr_t tmolVrendEia608Open(
   Int   *instance
)
```

Parameters

`instance`	Pointer (returned) to the instance.

Description

This function creates an instance and returns a pointer to the opened instance. The error value must be checked to decide whether the returned value is valid or not.

tmalVrendEia608Close

```
tmLibappErr_t tmalVrendEia608Close(
   Int   instance
)
```

Parameters

`instance`	The instance to be closed.

Description

This function closes an instance of the **VrendEia608** and the 2D library.

tmolVrendEia608Close

```
extern tmLibappErr_t tmolVrendEia608Close(
   Int   instance
)
```

Parameters

`instance`	The instance to be closed.

Description

This function closes an instance of the **VrendEia608** library.

tmalVrendEia608Start

```
tmLibappErr_t tmalVrendEia608Start(
   Int   instance
)
```

Parameters

`instance`	The instance.

Description

This function puts the instance into streaming mode. All input data is requested by the instance. This is implemented in a loop that calls the **datain** callback function to retrieve data from an upstream component. The decoder exits the loop when the corresponding **stop** is called. The **tmalVrendEia608DecodePacket** function is called to process incoming data. The a decision is made to possibly update the screen.

tmolVrendEia608Start

```
extern tmLibappErr_t tmolVrendEia608Start(
   Int   instance
)
```

Parameters

`instance`	The instance.

Description

This function puts the instance into streaming mode. All input data is requested by the instance. This is implemented in a loop that calls the **datain** callback function to retrieve data from an upstream component. The decoder exits the loop when the corresponding **stop** is called. The **tmalVrendEia608DecodePacket** function is called to process incoming data. The a decision is made to possibly update the screen.

The loop is runs in a separate task and is fed with incoming data via data queues and controlled via control queues.

tmalVrendEia608Stop

```
tmLibappErr_t tmalVrendEia608Stop(
   Int   instance
)
```

Parameters

`instance`	The instance.

Description

This function is the counterpart of the **tmalVrendEia608Start** function. Calling this function causes an exit out of the processing loop.

tmolVrendEia608Stop

```
extern tmLibappErr_t tmolVrendEia608Stop(
   Int   instance
)
```

Parameters

`instance`	The instance.

Description

This function is the counterpart of the **tmalVrendEia608Start** function. Calling this function causes an exit out of the processing loop. The decoding task is destroyed.

tmalVrendEia608GetCapabilities

```
tmLibappErr_t tmalVrendEia608GetCapabilities(
   ptmalVrendEia608Capabilities_t   *cap
)
```

Parameters

`cap`	Pointer to a variable in which to return a pointer to capabilities data.

Description

This function returns a pointer to the library-allocated capability structure.

tmolVrendEia608GetCapabilities

```
extern tmLibappErr_t tmolVrendEia608GetCapabilities(
   ptmolVrendEia608Capabilities_t   *cap
)
```

Parameters

`cap`	Pointer to a variable in which to return a pointer to capabilities data.

Description

This function returns a pointer to the library-allocated capability structure. This structure can be used in the **tsaDefaultInOutDescriptorCreate** function to establish a connection between two components.

tmolVrendEia608GetInstanceSetup

```
extern tmLibappErr_t tmolVrendEia608GetInstanceSetup(
   Int                                instance,
   ptmolVrendEia608InstanceSetup_t   *setup
)
```

Parameters

`instance`	The instance.
`setup`	Pointer to a variable in which to return a pointer to setup data.

Description

This function returns a pointer to the instance setup structure. The memory required for this structure is allocated by the library. The returned structure contains default values. The current setup of an instance can be retrieved by calling this function.

tmalVrendEia608InstanceConfig

```
tmLibappErr_t tmalVrendEia608InstanceConfig(
   Int                instance,
   ptsaControlArgs_t  args
)
```

Parameters

`instance`	The instance.
`args`	Control arguments.

Description

This function makes it possible to change component parameters when it is running. Certain parameters can be modified and status information can be retrieved. The EIA-608A decoder allows setting of the service and retrieval of the available service information.

tmolVrendEia608InstanceConfig

```
extern tmLibappErr_t tmolVrendEia608InstanceConfig(
   Int                  instance,
   UInt32               flags,
   ptsaControlArgs_t    args
)
```

Parameters

`instance`	The instance.
`flags`	Flags to control the behavior of this function. When reading information, it is important to pass the **tsaControlWait** flag.
`args`	Control arguments.

Description

This function makes it possible to change component parameters when it is running. Certain parameters can be modified and status information can be retrieved. The EIA-608A decoder allows setting of the service and retrieval of the available service information.

The control queues have to be set up. This is done by using the **tsaDefaultControl-DescriptorCreate** function.

tmalVrendEia608InstanceSetup

```
tmLibappErr_t tmalVrendEia608InstanceSetup(
   Int                                instance,
   tmalVrendEia608InstanceSetup_t   *setup
)
```

Parameters

instance	The instance.
setup	Pointer to instance setup data.

Description

This function initializes an instance of the EIA-608A library. The library-specific fields are copied. All information passed to the instance setup function can be thrown away after calling this function. Section **tmolVrendEia608InstanceSetup_t** describes the setup structure. A window manager virtual window is created, its dimensions dependent on the font size. The EIA-608A decoder is initialized.

tmolVrendEia608InstanceSetup

```
extern tmLibappErr_t tmolVrendEia608InstanceSetup(
   Int                                instance,
   tmolVrendEia608InstanceSetup_t   *setup
)
```

Parameters

instance	The instance.
setup	Pointer to instance setup data.

Description

This function initializes an instance of the EIA-608A library. The library-specific fields are copied. All information passed to the instance setup function can be thrown away after calling this function. Section **tmolVrendEia608InstanceSetup_t** describes the setup structure. A window manager virtual window is created, its dimensions dependent on the font size. The EIA-608A decoder instance is initialized.

tmalVrendEia608RedrawFunc

```
static tmLibappErr_t tmalVrendEia608RedrawFunc()
```

Description

This is the **VrendEia608** library internal callback function that is called by the window manager in case of an update condition. This function can be called in two different contexts. The first one is the **VrendEia608** context when the screen is to be updated because of a change in the EIA-608A buffer. The second one is the application context when something happens with the virtual window owned by the **VrendEia608** library (for instance, the clipping area has changed through display or hide of windows overlapping the EIA-608A window).

tmalVrendEia608DecodePacket

```
tmLibappErr_t tmalVrendEia608DecodePacket(
   Int              instance,
   tmAvPacket_t    *inpacket
)
```

Parameters

`instance`	Contains instance ID.
`inpacket`	Pointer to the input packet. The packet must contain four valid bytes in the data buffer. Byte 0 is the type. Byte 1 is 1 for valid and 0 for invalid data. Bytes 2 and 3 contain the EIA-608A data.

Description

This function decodes one input packet. The internal output of the decoder is stored in a character buffer. The content of this buffer is rendered in the context of the callback function.

Depending of the outcome of the interpretation, this function can force the window manager to redraw the EIA-608A related virtual window. Normally this function is called in the **tmalVrendEia608Start** function. If the application is working in data-push mode, this function is called directly. The decode packet function is called every frame to fulfill the EIA-608A requirements (even when no data is delivered from the upstream component). This means automatic switch off after 45 frames of no data and automatic activation after receiving 12 valid data packets.

tmalVrendEia608FieldVsync

```
extern tmLibappErr_t tmalVrendEia608FieldVsync()
```

Description

This function decides in conjunction with the **L21_Interpret_Data** function whether the screen needs to be updated or not. It is used to manage dynamic effects (e.g. flashing or scrolling).

VrendEia608 API Enumerations and Data Structures

This section presents the enumerations and data structures of the VrendEia608 API.

Name	Page
Eia608_Field_t	97
Eia608_Service_t	98
Eia608_XDSPackTypes_t	99
tmalVrendEia608ConfigTypes_t	102
tmalVrendEia608InstanceSetup_t	103
tmolVrendEia608InstanceSetup_t	105
Eia608_ATVEFPackTypes_t	107
tmVrendEia608ProgressVCHIP_t	108
tmVrendEia608ProgressXDS_t	109
tmVrendEia608ProgressATVEF_t	110

Eia608_Field_t

```
enum Eia608_Field_t {
   EIA608_FIELD1,
   EIA608_FIELD2
};
```

Fields

EIA608_FIELD1	First field of the TV frame.
EIA608_FIELD2	Second field of the TV frame.

Description

This enum defines keywords for field 1 and field 2 (see also **L21_Interpret_Data**).

Eia608_Service_t

```
enum Eia608_Service_t {
   EIA608_CC1,
   EIA608_CC2,
   EIA608_CC3,
   EIA608_CC4,
   EIA608_T1,
   EIA608_T2,
   EIA608_T3,
   EIA608_T4,
   EIA608_XDS,
   EIA608_ATVEF,
   EIA608_OFF,
   EIA608_UNKNOWN
};
```

Fields

EIA608_CC1	Closed Caption channel 1, field 1
EIA608_CC2	Closed Caption channel 2, field 1
EIA608_CC3	Closed Caption channel 1, field 2
EIA608_CC4	Closed Caption channel 2, field 2
EIA608_T1	Text channel 1, field 1
EIA608_T2	Text channel 2, field 1
EIA608_T3	Text channel 1, field 2
EIA608_T4	Text channel 2, field 2
EIA608_XDS	Extended Data Services
EIA608_ATVEF	ATVEF Data Services
EIA608_OFF	Decoder is switched off. Only the service detection is working (**L21_Is_Service_Present**).
EIA608_UNKNOWN	Only for internal use of the decoder. If the decoder is not synchronized to the Line 21 data stream, the internal service variable(s) (tL21_acq.iService) is (are) set to **EIA608_UNKNOWN**.

Description

Line 21 Data Service Data Channel and source definitions (see also **L21_Set_Mode**).

Eia608_XDSPackTypes_t

```
enum Eia608_XDSPackTypes_t {
   EIA608_NO_XDS_PACKAGE,
   EIA608_PROGRAM_ID_NO,
   EIA608_LENGTH,
   EIA608_PROGRAM_NAME,
   EIA608_PROGRAM_TYPE,
   EIA608_PROGRAM_RATING,
   EIA608_AUDIO_SERVICE,
   EIA608_CAPTION_SERVICE,
   EIA608_ASPECT_RATIO,
   EIA608_COMPOSITE_PACK1,  EIA608_COMPOSITE_PACK2,
   EIA608_PROG_DESCR_ROW1,  EIA608_PROG_DESCR_ROW2,
   EIA608_PROG_DESCR_ROW3,  EIA608_PROG_DESCR_ROW4,
   EIA608_PROG_DESCR_ROW5,  EIA608_PROG_DESCR_ROW6,
   EIA608_PROG_DESCR_ROW7,  EIA608_PROG_DESCR_ROW8,
   EIA608_F_PROGRAM_ID_NO,
   EIA608_F_LENGTH,
   EIA608_F_PROGRAM_NAME,
   EIA608_F_PROGRAM_TYPE,
   EIA608_F_PROGRAM_RATING,
   EIA608_F_AUDIO_SERVICE,
   EIA608_F_CAPTION_SERVICE,
   EIA608_F_ASPECT_RATIO,
   EIA608_F_COMPOSITE_PACK1,  EIA608_F_COMPOSITE_PACK2,
   EIA608_F_PROG_DESCR_ROW1,  EIA608_F_PROG_DESCR_ROW2,
   EIA608_F_PROG_DESCR_ROW3,  EIA608_F_PROG_DESCR_ROW4,
   EIA608_F_PROG_DESCR_ROW5,  EIA608_F_PROG_DESCR_ROW6,
   EIA608_F_PROG_DESCR_ROW7,  EIA608_F_PROG_DESCR_ROW8,
   EIA608_NETWORK_NAME,
   EIA608_CALL_LETTERS,
   EIA608_TAPE_DELAY,
   EIA608_TIME_OF_DAY,
   EIA608_CAPTURE_ID,
   EIA608_DATA_LOCATION,
   EIA608_LOCAL_TIME_ZONE,
   EIA608_OUT_BAND_CH_NO,
   EIA608_WEATHER_CODE,
   EIA608_WEATHER_MESSAGE
};
```

Fields

`EIA608_NO_XDS_PACKAGE`	No XDS package available.
`EIA608_PROGRAM_ID_NO`	Program identification number of the current program.

EIA608_LENGTH	Length of the current program.
EIA608_PROGRAM_NAME	Name of the current program.
EIA608_PROGRAM_TYPE	Type of the current program.
EIA608_PROGRAM_RATING	Program rating (V-Chip) of the current program.
EIA608_AUDIO_SERVICE	Audio service availability of the current program.
EIA608_CAPTION_SERVICE	Caption service availability of the current program.
EIA608_ASPECT_RATIO	Aspect ratio of the current program.
EIA608_COMPOSITE_PACK1	First composite package of the current program.
EIA608_COMPOSITE_PACK2	Second composite package of the current program.
EIA608_PROG_DESCR_ROW1	First program description row of the current program.
EIA608_PROG_DESCR_ROW2	Second program description row of the current program.
EIA608_PROG_DESCR_ROW3	Third program description row of the current program.
EIA608_PROG_DESCR_ROW4	Fourth program description row of the current program.
EIA608_PROG_DESCR_ROW5	Fifth program description row of the current program.
EIA608_PROG_DESCR_ROW6	Sixth program description row of the current program.
EIA608_PROG_DESCR_ROW7	Seventh program description row of the current program.
EIA608_PROG_DESCR_ROW8	Eighth program description row of the current program.
EIA608_F_PROGRAM_ID_NO	Program identification number of the future program.
EIA608_F_LENGTH	Length of the future program.
EIA608_F_PROGRAM_NAME	Name of the future program.
EIA608_F_PROGRAM_TYPE	Type of the future program.
EIA608_F_PROGRAM_RATING	Program rating (V-Chip) of the future program.
EIA608_F_AUDIO_SERVICE	Audio service availability of the future program.
EIA608_F_CAPTION_SERVICE	Caption service availability of the future program.
EIA608_F_ASPECT_RATIO	Aspect ratio of the future program.
EIA608_F_COMPOSITE_PACK1	First composite package of the future program.
EIA608_F_COMPOSITE_PACK2	Second composite package of the future program.
EIA608_F_PROG_DESCR_ROW1	First program description row of the future program.

EIA608_F_PROG_DESCR_ROW2	Second program description row of the future program.
EIA608_F_PROG_DESCR_ROW3	Third program description row of the future program.
EIA608_F_PROG_DESCR_ROW4	Fourth program description row of the future program.
EIA608_F_PROG_DESCR_ROW5	Fifth program description row of the future program.
EIA608_F_PROG_DESCR_ROW6	Sixth program description row of the future program.
EIA608_F_PROG_DESCR_ROW7	Seventh program description row of the future program.
EIA608_F_PROG_DESCR_ROW8	Eighth program description row of the future program.
EIA608_NETWORK_NAME	Name of the network.
EIA608_CALL_LETTERS	Call letters of the broadcaster.
EIA608_TAPE_DELAY	Delay of the tape.
EIA608_TIME_OF_DAY	Time of the day.
EIA608_CAPTURE_ID	Capture identification.
EIA608_DATA_LOCATION	Data location.
EIA608_LOCAL_TIME_ZONE	Local time zone.
EIA608_OUT_BAND_CH_NO	Out of band channel number.
EIA608_WEATHER_CODE	Weather code.
EIA608_WEATHER_MESSAGE	Weather message.

Description

Definition of XDS types (see also **L21_Set_XDSfunc**)

tmalVrendEia608ConfigTypes_t

```
enum tmalVrendEia608ConfigTypes_t {
   VRENDEIA608_CONFIG_SET_SERVICE,
   VRENDEIA608_CONFIG_IS_SERVICE_PRESENT
};
```

Fields

`VRENDEIA608_CONFIG_SET_SERVICE` Selects a new service or switches the decoder off.

`VRENDEIA608_CONFIG_IS_SERVICE_PRESENT`
Retrieves the information on service availability.

Description

Definition of flags for the configuration functions. See also **tmalVrendEia608Instance-Config** and **tmolVrendEia608InstanceConfig**.

tmalVrendEia608InstanceSetup_t

```
typedef struct {
   struct defaultSetup;
   struct plainFont;
   struct italicFont;
   struct xOffset;
   struct yOffset;
   struct pBackPlane;
   struct wmStackingOrder;
   struct frameRate;
   struct colorKeyY;
   struct colorKeyU;
   struct colorKeyV;
   struct textModeHeight;
   struct service;
} tmalVrendEia608InstanceSetup_t;
```

Fields

`defaultSetup`	This field points to the default setup structure that contains a.o. the callback functions. The chapter TriMedia Software Architecture has more information about default structures
`plainFont`	This field points to an pre-created font structure that contains the plain font. It has no default value. It must be initialized during the setup procedure. Creation of a font can be accomplished by opening the 2D library, creating a font and closing the 2D library.
`italicFont`	This field points to an pre-created font structure that contains the italic font. It has no default value. It must be initialized during the setup procedure.
`xOffset`	This field contains the horizontal offset at which the library will paint in the virtual window. Since the window has an offset itself, the default value is zero.
`yOffset`	This field contains the vertical offset at which the library will paint in the virtual window. Since the window has an offset itself, the default value is zero.

`pBackPlane`	This field contains the pointer to the back plane packet. The VrendEia608 decoder draws in this back plane. Since the decoder window is a virtual window, all 2D library calls manipulate this buffer directly. The application has to provide a back plane with a proper size.
`wmStackingOrder`	This field specifies the stacking order of the EIA-608 decoder window. The default is **wmSO_ALWAYS_ON_BOTTOM**.
`frameRate`	This field specifies the frame rate of the displayed video. It is used to implement flashing that will be on/off for half second intervals. Default value is 30.
`colorKeyY`	This field specifies the luminance of the background color, the color that is 100 percent transparent. The default is 0x00.
`colorKeyU`	This field specifies the chrominance of the background color - the color that is 100 percent transparent. The default is 0x20.
`colorKeyV`	This field specifies the chrominance of the background color, the color that is displayed 100 percent transparent. The default is 0x20.
`textModeHeight`	This field specifies the height of the box if the user changes to one of the text services. Default is eight rows.
`service`	This field specifies what service is active after start-up. The default is EIA608_CC1.
`pATVEFbuffer`	Points to ATVEF buffer. If buffer pointer is Null, no ATVEF service will be provided. Default is Null.
ulATVEFlength	Size of the ATVEF buffer. This value will be ignored if pointer to buffer is Null.

Description

Definition of the instance setup structure. All fields have to be initialized by the application during setup.

tmolVrendEia608InstanceSetup_t

```
typedef struct {
   ptsaDefaultInstanceSetup_t defaultSetup;
   ptsa2DFont_t               plainFont;
   ptsa2DFont_t               italicFont;
   UInt32                     xOffset;
   UInt32                     yOffset;
   ptmAvPacket_t              pBackPlane;
   tsaWMStackingOrder_t       wmStackingOrder;
   UInt8                      frameRate;
   UInt8                      colorKeyY;
   UInt8                      colorKeyU;
   UInt8                      colorKeyV;
   UInt32                     textModeHeight;
   UInt32                     service;
   Int8                      *pATVEFbuffer;
   UInt32                     ulATVEFlength;
} tmolVrendEia608InstanceSetup_t;
```

Fields

`defaultSetup`	This field points to the default setup structure that contains a.o. the callback functions. For more information on default structures, see TSSA documentation.
`plainFont`	This field points to an pre-created font structure that contains the plain font. It has no default value. It must be initialized during the setup procedure. Creation of a font can be accomplished by opening the 2D library, creating a font and closing the 2D library.
`italicFont`	This field points to an pre-created font structure that contains the italic font. It has no default value. It must be initialized during the setup procedure.
`xOffset`	This field contains the horizontal offset at which the library will paint in the virtual window. Since the window has an offset itself, the default value is zero.
`yOffset`	This field contains the vertical offset at which the library will paint in the virtual window. Since the window has an offset itself, the default value is zero.
`pBackPlane`	This field contains the pointer to the back plane packet. The VrendEia608 decoder draws in this back plane. Since the decoder window is a virtual

	window, all 2D library calls manipulate this buffer directly. The application has to provide a back plane with a proper size.
`wmStackingOrder`	This field specifies the stacking order of the EIA-608 decoder window. The default value is **wmSO_ALWAYS_ON_BOTTOM**.
`frameRate`	This field specifies the frame rate of the displayed video. It is used to implement flashing that will be on/off for half second intervals. Default value is 30.
`colorKeyY`	This field specifies the luminance of the background color - the color that is 100 percent transparent. The default is 0x00.
`colorKeyU`	This field specifies the chrominance of the background color - the color that is 100 percent transparent. The default is 0x20.
`colorKeyV`	This field specifies the chrominance of the background color - the color that is 100 percent transparent. The default is 0x20.
`textModeHeight`	This field specifies the height of the box if the user changes to one of the text services. Default is eight rows.
`service`	This field specifies what service is active after start-up. The default is EIA608_CC1.
`pATVEFbuffer`	Points to ATVEF buffer. If buffer pointer is Null, no ATVEF service will be provided. Default is Null.
`ulATVEFlength`	Size of the ATVEF buffer. This value will be ignored if pointer to buffer is Null.

Description

Definition of the instance setup structure. A pointer to this structure is returned by the **tmolVrendEia608GetInstanceSetup** function. At least three fields (**plainFont**, **italicFont**, **wmInstance**) have to be initialized by the application. The other fields contain default values.

Eia608_ATVEFPackTypes_t

```
enum Eia608_ATVEFPackTypes_t {
   EIA608_ATVEF_RECEIVED,
   EIA608_ATVEF_BUFFER_FULL
};
```

Fields

`EIA608_ATVEF_RECEIVED`	Everything was working properly and the URL string is located in **pBuffer** of the **tmVrendEia608-ProgressXDS_t** structure.
`EIA608_ATVEF_BUFFER_FULL`	The provided buffer was not big enough resulting in a overflow. The string located in **pBuffer** is invalid.

Description

Those flags are part of the codes that the progress function sends to the application. These flags specify how the ATVEF-URL has been sent out by the decoder.

tmVrendEia608ProgressVCHIP_t

```
typedef struct {
   UInt8          *pBuffer;
   Bool            validTimeStamp;
   tmTimeStamp_t   time;
} tmVrendEia608ProgressVCHIP_t, *ptmVrendEia608ProgressVCHIP_t;
```

Fields

`pBuffer`	Pointer to the two bytes, which have been decoded by the VrendEia608 decoder. No buffer size is necessary, since VCHIP data is always two bytes.
`validTimeStamp`	Flag that tells the application that the **time** structure is valid.
`time`	Contains time stamp. Is only valid if **validTimeStamp** is True.

Description

Structure sent out by the **progressFunc** callback. The application can determine the type of the received structure from the progress flags. **VrendEia608_Progress_VCHIP** indicates a Vchip progress report.

tmVrendEia608ProgressXDS_t

```
typedef struct {
   tEia608_XDSPackTypes   type;
   UInt8                  numBytes;
   UInt8                 *pBuffer;
   Bool                   validTimeStamp;
   tmTimeStamp_t          time;
} tmVrendEia608ProgressXDS_t, *ptmVrendEia608ProgressXDS_t;
```

Fields

`type`	Flags that describes the type of XDS service received and how to interpret the received buffer.
`numBytes`	Contains number of valid Bytes in the buffer pointed to by pBuffer.
`pBuffer`	Pointer to the decoded bytes.
`validTimeStamp`	Flag that tells the application that the time structure is valid.
`time`	Contains time stamp. Is only valid if **validTimeStamp** is True.

Description

Structure sent out by the **progressFunc** callback. The application can determine the type of the received structure from the progress flags. **VrendEia608_Progress_XDS** indicates an XDS progress report.

tmVrendEia608ProgressATVEF_t

```
typedef struct {
   tEia608_XDSPackTypes   type;
   UInt8                  numBytes;
   UInt8                 *pBuffer;
   Bool                   validTimeStamp;
   tmTimeStamp_t          time;
} tmVrendEia608ProgressATVEF_t, *ptmVrendEia608ProgressATVEF_t;
```

Fields

`type`	Flags that describes the type of ATVEF service received and how to interpret the received buffer.
`numBytes`	Number of valid bytes in ***pBuffer**.
`pBuffer`	Pointer to the decoded bytes. The received string is not terminated by a null byte.
`validTimeStamp`	Flag that tells the application that the time structure is valid.
`time`	Contains time stamp. Is only valid if **validTimeStamp** is True.

Description

Structure sent out by the **progressFunc** callback. The application can determine the type of the received structure from the progress flags. **VrendEia608_Progress_ATVEF** indicates an ATVEF progress report.

Chapter 4

Closed-Captioning (EIA-708) API

Topic	Page
DTVCC Decoder (EIA-708) Overview	112
DTVCC Decoder (EIA-708) Inputs and Outputs	112
DTVCC Decoder (EIA-708) Progress	113
DTVCC Decoder (EIA-708) Error	114
DTVCC Decoder (EIA-708) API Data Structures	114
DTVCC Decoder (EIA-708) API Functions	125

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

DTVCC Decoder (EIA-708) Overview

DTVCC (Digital Television Closed Caption) is a migration of the closed-captioning concepts and capabilities developed in the 1970´s for the NTSC television video signals to the high-definition television environment defined by the ATV (Advanced Television) Grand Alliance and standardized by the ATSC (Advanced Television Systems Committee). This new television environment provides for larger screens and higher screen resolutions, and higher data rates for transmission of closed-captioning data.

NTSC Closed Caption consist of an analog waveform inserted into Line 21 of the NTSC Vertical Blanking Interval (VBI). This waveform provides a transport channel which can deliver 2 bytes of data on every field of video. This translates to 120 bytes per second, or 960 bits per second (bps). In contrast, ATV Closed Captioning is transported as a logical data channel in the ATV digital bit stream. Of the ATV bitstream bit rate (which is 19.4 Mbps for terrestrial broadcast, and 38.4 Mbps for cable), ATV-specifies that closed captioning is allocated 9600 bps. This increased capacity opens the possibilities for the simultaneous transmission of captions in multiple languages and at multiple reading levels.

The ATV standard boasts an increased screen resolution range 480, 720 or 1080 active scan lines, vs. 525 scan lines for NTSC. These added resolution of 720 or 1080 lines provide for more defined representations of character fonts and other on-screen objects. The heart of DTVCC caption display is the caption "window" which is identical to the window concept found in all computer Graphical User Interfaces (GUI). Windows are placed within the ATV screen, and caption text is placed within windows. Windows and text have a variety of color, size and other attributes.

Background

This document assumes that the reader is familiar with the concepts of TSSA as documented in Book 3, *Software Architecture*, Part B.

DTVCC Decoder (EIA-708) Inputs and Outputs

The DTVCC Decoder (EIA-708) expects its inputs data via a standard TSSA queue. The decoder output is a updated video packet. The update is done using the window manager and the 2D library.

The decoder does not support multiple instances.

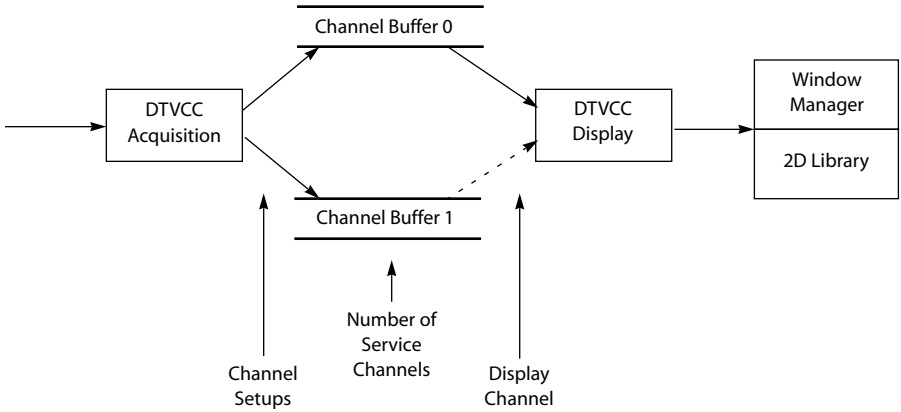
Compliance With the DTVCC Standard

The DTVCC decoder supports the following EIA-708 features:

- All EIA-708 (Chapter. 8) 'minimum decoder' features with single fixed-space font (except for center and right text justification).
- Maximum of one 7 row visible window or four single row visible windows on the display at any one time.
- Automatic word-wrapping in fixed size windows
- All 64 foreground/background colors

Multiple Service Channel Decoding

A single decoder instance supports multiple service channel decoding. This feature allows to switch from one service channel to another without delay.



The application controls the multiple channel decoding mainly with three settings.

Channel setup	Determines which service should be decoded in which channel buffer.
Number of service channels	Determines how many services can be decoded in parallel.
Display channel	Determines which channel should be displayed.

See also **tmolVrendEia708InstanceSetup** on page 129 and **tmolVrendEia708InstanceConfig** on page 131.

DTVCC Decoder (EIA-708) Progress

The DTVCC Decoder does not support progress functions.

DTVCC Decoder (EIA-708) Error

The DTVCC Decoder (EIA-708) reports errors to the error function, if the soft errors are activated, see **tmolVrendEia708InstanceSetup_t** on page 116.

Error codes

`VRENDEIA708_ERR_DTVCC_PKT_LEN`	The received DTVCC packet has an invalid length.
`VRENDEIA708_ERR_NO_SERV_DATA`	Their are no data for the selected service.
`VRENDEIA708_ERR_DATA_INCONSISTENCY`	The received DTVCC data are inconsistent.

DTVCC Decoder (EIA-708) API Data Structures

This section presents the DTVCC Decoder (EIA-708) data structures.

Name	Page
tmolVrendEia708Capabilities_t	115
tmalVrendEia708Capabilities_t	115
tmolVrendEia708InstanceSetup_t	116
tmalVrendEia708InstanceSetup_t	116
tmVrendEia708Fonts_t	118
tmVrendEia708FontStyles_t	119
tmVrendEia708AR_t	120
tmVrendEia708ServDecSetup_t	120

tmolVrendEia708Capabilities_t

```
typedef tmalVrendEia708Capabilities_t
   tmolVrendEia708Capabilities_t,
   *ptmolVrendEia708Capabilities_t;
```

tmalVrendEia708Capabilities_t

```
typedef struct {
   ptsaDefaultCapabilities_t  defaultCapabilities;
} tmalVrendEia708Capabilities_t, *ptmalVrendEia708Capabilities_t;
```

Fields

`defaultCapabilities`	TSA default capabilities

Description

This structure passes the DTVCC Decoder (EIA-708) capabilities to the application.

tmalVrendEia708InstanceSetup_t

```
typedef struct {
   ptsaDefaultInstanceSetup_t   defaultSetup;
   Int                          gfxInstance;
   ptmVrendEia708Fonts_t        fontRefs;
   tsaYUVColor_t                colorKey;
   Int                          wmInstance;
   ptmVideoFormat_t             displayFormat;
   Int                          safeAreaScaleX;
   Int                          safeAreaScaleY;
   Int                          safeAreaOffsetX;
   Int                          safeAreaOffsetY;
   UInt                         timeOutDelay;
   UInt                         frameRate;
   Bool                         reportSoftErrors;
   Bool                         autoResyncOnStreamErr;
   tmVrendEia708AR_t            dispAspectRatio;
   Bool                         displayActive;
   Int                          numServDecChannels;
   Int                          displayChannel;
   tmVrendEia708ServDecSetup_t  chanSetups[2];
   Bool                         servDataInputTestMode;
} tmalVrendEia708InstanceSetup_t, *ptmalVrendEia708InstanceSetup_t;
```

tmolVrendEia708InstanceSetup_t

```
typedef tmalVrendEia708InstanceSetup_t
   tmolVrendEia708InstanceSetup_t,
   *ptmolVrendEia708InstanceSetup_t;
```

Fields

`defaultSetup`	For compliance with TSA, this is a pointer to structure of the standard type.
`gfxInstance`	2D library instance (required for output)
`fontRefs`	2D font reference
`colorKey`	Transparent Color
`wmInstance`	Window Manger instance (required for output)
`displayFormat`	Format of WM BackPlane pkt

`safeAreaScaleX`
`safeAreaScaleY`
`safeAreaOffsetX`
`safeAreaOffsetY`
	These setup parameters (safeAreaScaleX, safeAreaScaleY, safeAreaOffsetX and safeAreaOffsetY) are for adjusting the safe area scale and offset.
	Nominal safe area scale and offset is set to the center 90% of the size of the WM Backplane packet referenced by displayFormat (above).
	1) The safeAreaScale is expressed as a % of displayFormat pkt size.
	2) The safeAreaOffset is expressed as a % of the combined nominal safe area and the above scale setting.
	3) If safeAreaScale is set to a value ≥ +10%, then the safe area will occupy the whole of the displayFormat pkt size so safeAreaOffset settings will have no effect.

`timeOutDelay`	Defines the time in seconds (default 1 min.) between the last received packet on a decoder channel and the automatic de-activation of that channel. If the channel is currently being displayed then the display is also cleared.
`frameRate`	Picture frame rate in frames/second.
`reportSoftErrors`	Report non-fatal (soft) stream errors.
`autoResyncOnStreamErr`	Re-sync when possible. stream corruption.
`dispAspectRatio`	Aspect Ratio of display device.
`displayActive`	Initial display state on/off.
`numServDecChannels`	Number of channels which should be simultaneously decoded (1 or 2).
`displayChannel`	Initial decoder display channel.
`chanSetups[2]`	Initial settings for the channels.
`servDataInputTestMode`	TESTING ONLY: Set input data stream to be DTVCC service (block) data (not DTVCC packets).

Description

This structure passes the instance setup to the DTVCC Decoder (EIA-708).

tmVrendEia708Fonts_t

```
typedef struct {
   ptmVrendEia708FontStyles_t  largePlain;
   ptmVrendEia708FontStyles_t  largeItalic;
   ptmVrendEia708FontStyles_t  stdPlain;
   ptmVrendEia708FontStyles_t  stdItalic;
   ptmVrendEia708FontStyles_t  smallPlain;
   ptmVrendEia708FontStyles_t  smallItalic;
} tmVrendEia708Fonts_t, *ptmVrendEia708Fonts_t;
```

Fields

`largePlain`	Large plain fonts.
`largeItalic`	Large Italic fonts.
`stdPlain`	Standard size plain fonts.
`stdItalic`	Standard size italic fonts.
`smallPlain`	Small Plain fonts.
`smallItalic`	Small Italic fonts.

Description

This structure is used to define the various EIA-708 fonts sizes.

Note
Only standard sized (Plain/Italic) fonts are currently supported.

tmVrendEia708FontStyles_t

```
typedef struct {
   ptsa2DFont_t   defaultStyle;
   ptsa2DFont_t   monoSerif;
   ptsa2DFont_t   propSerif;
   ptsa2DFont_t   mono;
   ptsa2DFont_t   prop;
   ptsa2DFont_t   casual;
   ptsa2DFont_t   cursive;
   ptsa2DFont_t   smallCaps;
} tmVrendEia708FontStyles_t, *ptmVrendEia708FontStyles_t;
```

Fields

`defaultStyle`	Default font.
`monoSerif`	Font in mono-spaced serif style.
`propSerif`	Font in proportionally spaced serif style.
`mono`	Font in mono-spaced style.
`prop`	Font in proportionally spaced style.
`casual`	Font in casual style.
`cursive`	Font in cursive style.
`smallCaps`	Font in small caps style.

Description

This structure defines the various EIA-708 font styles.

Note
Only default style is currently supported.

tmVrendEia708AR_t

```
typedef enum {
   VRENDEIA708_AR_4TO3   = 0,
   VRENDEIA708_AR_16TO9  = 1
} tmVrendEia708AR_t, *ptmVrendEia708AR_t;
```

Values

VRENDEIA708_AR_4TO3	The Aspect Ratio is 4 to 3.
VRENDEIA708_AR_16TO9	The Aspect Ratio is 16 to 9.

Description

This enumerates aspect ratios of the display as well as the decoder channels.

tmVrendEia708ServDecSetup_t

```
typedef struct {
   Bool               chanActive;
   Int                servNum;
   tmVrendEia708AR_t  srcAspectRatio;
} tmVrendEia708ServDecSetup_t, *ptmVrendEia708ServDecSetup_t;
```

Fields

channActive	Activates/deactivates decoding.
servNum	Caption service (1–6).
srcAspectRatio	Caption Aspect Ratio.

Description

This structure defines the settings for an individual channel.

tmVrendEia708ConfigCommands_t

```
typedef enum {
   VRENDEIA708_CONFIG_SET_SERV_NUM        = tsaCmdUserBase + 0x41,
   VRENDEIA708_CONFIG_SET_SOURCE_AR       = tsaCmdUserBase + 0x42,
   VRENDEIA708_CONFIG_SET_CHANNEL_ON      = tsaCmdUserBase + 0x43,
   VRENDEIA708_CONFIG_SET_CHANNEL_OFF     = tsaCmdUserBase + 0x44,
   VRENDEIA708_CONFIG_SET_DISPLAY_CHANNEL = tsaCmdUserBase + 0x45,
   VRENDEIA708_CONFIG_SET_DISPLAY_ON      = tsaCmdUserBase + 0x46,
   VRENDEIA708_CONFIG_SET_DISPLAY_OFF     = tsaCmdUserBase + 0x47,
   VRENDEIA708_CONFIG_REFRESH_DISPLAY     = tsaCmdUserBase + 0x48,
   VRENDEIA708_CONFIG_SET_FRAME_RATE      = tsaCmdUserBase + 0x49,
   VRENDEIA708_CONFIG_RESET_CHANNELS      = tsaCmdUserBase + 0x4a,
   VRENDEIA708_CONFIG_GET_SERV_NUM        = tsaCmdUserBase + 0x81,
   VRENDEIA708_CONFIG_GET_SOURCE_AR       = tsaCmdUserBase + 0x82,
   VRENDEIA708_CONFIG_GET_CHANNEL_ACTIVE  = tsaCmdUserBase + 0x83,
   VRENDEIA708_CONFIG_GET_DISPLAY_INFO    = tsaCmdUserBase + 0x84,
   VRENDEIA708_CONFIG_GET_FRAME_RATE      = tsaCmdUserBase + 0x85,
} tmVrendEia708ConfigCommands_t, *ptmVrendEia708ConfigCommands_t;
```

Values

`VRENDEIA708_CONFIG_SET_SERV_NUM` Sets caption service number to be decoded (selected from available list in PMT) on specified service decoder channel (relevant service data extracted from supplied DTVCC packet data).
Input Parameters: servDecChannel, p.servNum
Output Parameters: none

`VRENDEIA708_CONFIG_GET_SERV_NUM` Input Parameters: servDecChannel
Output Parameters: p.servNum

`VRENDEIA708_CONFIG_SET_SOURCE_AR`
Sets intended aspect ratio for caption service being decoded on specified service decoder channel (from PMT data).

Input Parameters: **servDecChannel**, **p.srcAspectRatio**

Output Parameters: none

`VRENDEIA708_CONFIG_GET_SOURCE_AR`
Input Parameters: **servDecChannel**
Output Parameters: **p.srcAspectRatio**

`VRENDEIA708_CONFIG_SET_CHANNEL_ON`
Turns *specified* caption channel ON (causes decoding from selected DTVCC service data to be performed on specified channel).

Input Parameters: **servDecChannel**
Output Parameters: none

`VRENDEIA708_CONFIG_SET_CHANNEL_OFF`

Turns *specified* caption channel OFF (causes decoding to be disabled on specified channel).

Input Parameters: **servDecChannel**
Output Parameters: none

`VRENDEIA708_CONFIG_GET_CHANNEL_ACTIVE`

Input Parameters: **servDecChannel**
Output Parameters: **p.chanActive**

`VRENDEIA708_CONFIG_SET_DISPLAY_CHANNEL`

Sets which service channel owns the output display.

Input Parameters: **servDecChannel**
Output Parameters: none

`VRENDEIA708_CONFIG_SET_DISPLAY_ON`

Turns *current* caption display ON (causes output display to be rendered according to decoded/stored caption data for current channel).

Input Parameters: none
Output Parameters: none

`VRENDEIA708_CONFIG_SET_DISPLAY_OFF`

Turns *current* caption display OFF (clears all active windows, decoding and storage of caption data continues but output display is not rendered).

Input Parameters: none
Output Parameters: none

`VRENDEIA708_CONFIG_GET_DISPLAY_INFO`

Input Parameters: none
Output Parameters: **servDecChannel**, **p.displayActive**

`VRENDEIA708_CONFIG_SET_FRAME_RATE`

Sets the frame rate in frames/second. Some dynamic effects e.g. the flash frequency is derived from the frame rate.

Input Parameters: **p.frameRate**
Output Parameters: none

`VRENDEIA708_CONFIG_GET_FRAME_RATE`

Input Parameters: none
Output Parameters: **p.frameRate**

`VRENDEIA708_CONFIG_REFRESH_DISPLAY`

Refreshes *current* output display (clears and then redraws all active windows according to decoded/stored caption data for current channel).

Input Parameters: none
Output Parameters: none

`VRENDEIA708_CONFIG_RESET_CHANNELS`

Resets all decoder channels and clears the display.
Input Parameters: none
Output Parameters: none

Description

This enum is used in **tmolVrendEia708InstanceConfig** on page 131. The input and output parameter (**p.***xxxx*) are described in **tmVrendEia708ConfigParams_t** on page 124.

tmVrendEia708ConfigParams_t

```
typedef struct {
   Int   servDecChannel;
   union {
      Int                servNum;
      tmVrendEia708AR_t  srcAspectRatio;
      Bool               chanActive;
      Bool               displayActive;
      UInt               frameRate;
   } p;
} tmVrendEia708ConfigParams_t, *ptmVrendEia708ConfigParams_t;
```

Fields

`servDecChannel`	Service decoder channel number.
`servNum`	Caption service (1–6).
`srcAspectRatio`	Caption Aspect Ratio.
`chanActive`	Set the channel to active or inactive.
`displayActive`	Set the display to active or inactive.
`frameRate`	Define the frame rate of the video.

Description

This structure is used to pass additional parameters to **tmolVrendEia708InstanceConfig** on page 131.

DTVCC Decoder (EIA-708) API Functions

This section contains the DTVCC Decoder API function description.

Name	Page
tmolVrendEia708GetCapabilities	126
tmolVrendEia708Open	127
tmolVrendEia708Close	127
tmolVrendEia708GetInstanceSetup	128
tmolVrendEia708InstanceSetup	129
tmolVrendEia708Start	130
tmolVrendEia708Stop	130
tmolVrendEia708InstanceConfig	131
tmolVrendEia708FieldVsync	132

tmolVrendEia708GetCapabilities

```
tmLibappErr_t tmalVrendEia708GetCapabilities(
   ptmalVrendEia708Capabilities_t *pCap
);
```

Parameters

pCap	Pointer to a variable in which to return a pointer to the capabilities data.

Return Codes

TMLIBAPP_OK	Success.

Description

Gets the capabilities of the DTVCC Decoder (EIA-708).

tmolVrendEia708Open

```
tmLibappErr_t tmalVrendEia708Open(
   Int   *instance
);
```

Parameters

`instance`	Address of an integer that will hold the instance value for this DTVCC Decoder (EIA-708).

Return Codes

`TMLIBAPP_OK`	Success.

Description

The open function creates an instance of the DTVCC Decoder (EIA-708) and informs the user of its instance ID. The DTVCC Decoder (EIA-708) does not support multiple instances.

tmolVrendEia708Close

```
tmLibappErr_t tmalVrendEia708Close(
   Int   instance
);
```

Parameters

`instance`	The instance, as opened by **tmolVrendEia708Open**.

Return Codes

`TMLIBAPP_OK`	Success.
other	See **tsaClockClose**.

Description

Closes a stopped instance of the DTVCC Decoder (EIA-708).

tmolVrendEia708GetInstanceSetup

```
tmLibappErr_t tmolVrendEia708GetInstanceSetup(
   Int                                 instance,
   ptmolVrendEia708InstanceSetup_t   *setup
);
```

Parameters

`instance`	Instance previously opened by **tmolVrendEia708Open**.
`setup`	Pointer to a DTVCC Decoder (EIA-708) setup data structure, see **tmolVrendEia708InstanceSetup_t** on page 116.

Return Codes

`TMLIBAPP_OK`	Success.

Description

This function is used during initialization of the decoder. It returns the default settings for the decoder instance. The setup can then be further initialized by the application which normally is filling all the queues and the progress and error functions and then passed to **tmolVrendEia708InstanceSetup**.

tmolVrendEia708InstanceSetup

```
tmLibappErr_t tmalVrendEia708InstanceSetup(
   Int                              instance,
   tmalVrendEia708InstanceSetup_t *setup
);
```

Parameters

instance	**Instance previously opened by 'tmolVrendEia708Open'**
setup	Pointer to a DTVCC Decoder (EIA-708) setup data structure, see **tmolVrendEia708InstanceSetup_t** on page 116.

Return Codes

TMLIBAPP_OK	Success
VRENDEIA708_ERR_SETUP_SERV_NUM	The service number is invalid (legal values are from 1 to 6).
VRENDEIA708_ERR_SETUP_SRC_AR_TYPE	The aspect ratio of a source channel is wrong, see tmVrendEia708AR_t on page 120.
VRENDEIA708_ERR_SETUP_DISP_AR_TYPE	The aspect ratio of the display is wrong, see tmVrendEia708AR_t' on page 120.
VRENDEIA708_ERR_SETUP_DISPLAY_CHAN	The display channel is invalid. Valid channels are 1 and 2.
VRENDEIA708_ERR_SETUP_NUM_SERV_CHANS	The number of service channels is invalid. A valid number is 1 and 2.
VRENDEIA708_ERR_INTERNAL	An internal error has occurred.
TMLIBAPP_ERR_MEMALLOC_FAILED	The required memory could not be allocated.

Description

The instance previously opened by **tmolVrendEia708Open** is set up. Memory is allocated to store run-time instance data. The instance is marked as setup. **tmolVrendEia708Setup** should be called only once for each instance.

tmolVrendEia708Start

```
tmLibappErr_t tmalVrendEia708Start(
   Int instance
);
```

Parameters

`instance`	Instance previously opened by **tmolVrendEia708Open**.

Return Codes

`TMLIBAPP_OK`	Success.

Description

The DTVCC Decoder (EIA-708) represented by its instance ID is started. An independent task to execute the decoder code is started.

tmolVrendEia708Stop

```
tmLibappErr_t tmalVrendEia708Stop(
   Int instance
);
```

Parameters

`instance`	Instance previously opened by **tmolVrendEia708Open**.

Return Codes

`TMLIBAPP_OK`	Success.

Description

The DTVCC Decoder (EIA-708) represented by the instance ID is stopped. All packets held by the decoder are returned to their respective queues, and the decoder exits its processing loop in accordance with standard TSSA guidelines.

tmolVrendEia708InstanceConfig

```
tmLibappErr_t tmalVrendEia708InstanceConfig(
   Int                instance,
   ptsaControlArgs_t  args
);
```

Parameters

`instance`	Instance previously opened by **tmolVrendEia708Open**.
`args`	Pointer to a control structure (TSSA) used to modify the operation of the DTVCC Decoder (EIA-708).

Return Codes

`TMLIBAPP_OK`	Success.

The following codes are returned via **args–>retval**.

`VRENDEIA708_ERR_CONFIG_SERV_CHAN_NUM`	The service channel number is invalid. Valid numbers are 1 or 2.
`VRENDEIA708_ERR_CONFIG_SRC_AR_TYPE`	The aspect ratio of the source channel is invalid, see **tmVrendEia708AR_t** on page 120.
`VRENDEIA708_ERR_CONFIG_CODE`	The configuration command is invalid, see **tmVrendEia708ConfigCommands_t** on page 121.
`VRENDEIA708_ERR_CONFIG_SERV_NUM`	The display service number is invalid. Valid service numbers are 1–6.

Description

This function prepares a command to be sent to the DTVCC Decoder (EIA-708) task, which then synchronously reacts on it. The command is sent with default priority. The command (see **tmVrendEia708ConfigCommands_t** on page 121) is passed in the command field of the args structure. A possible return value is returned via the parameter field of the args structure.

tmolVrendEia708FieldVsync

```
tmLibappErr_t tmalVrendEia708FieldVsync(
   Int     instance,
   Bool    evenField,
   Bool   *screenNeedsUpdate
);
```

Parameters

`instance`	Instance, previously opened by **tmolVrendEia708-Open**.
`evenField`	Current displayed video field.
`screenNeedsUpdate`	Request to update the whole screen.

Return Codes

`TMLIBAPP_OK`	Success

Description

Some of the DTVCC features have a dynamic behavior. This function is used to synchronize the dynamic DTVCC effects of the video display.

It needs to be called once a field. A call during the VBI (Vertical Blank Interval) is recommended.

Chapter 5

HTML Parser (HtmlParser) API

Topic	Page
Overview	134
HTML Data Structures	141
HTML Enumerated Types	157
HTML API Data Structures	160
HTML API Functions	165
HTML Tags Supported	174

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

Overview

The TriMedia HTML Parser library gives your application the ability to parse HTML (hypertext markup language) which can then be passed to the HTML renderer library for display. The library is based on the HTML 3.2 standard[1] and complies with the TriMedia Software Architecture (TSA).

In addition to the HTML 3.2 standard, the parser provides several extensions.

- In INPUT tag, in addition to the input type of text fields, radio buttons, check boxes, etc., an input type of button can beused as a general purpose button such as push button and toggle button.
- An HSLIDER (HTML tag for horizontal slider) makes it possible to specify sliders (such as volume control bar) in HTML pages.
- The parser supports transparent background for the HTML document body. A "transparent" color (RGB color values) is chosen in the bgcolor attribute of the BODY tag. This "transparent" color (in tsa2D color) is also needed to set the transparent field in the HTML renderer setup structure. See the HTML Renderer API document for more information.

This is the HTML parser intended for use in a stand-alone DTV system. HTML pages can either be stored statically in a database or generated dynamically from the applications.

Currently, the HtmlParser does not support re-entrancy. Re-entrancy and other issues will be addressed in a future release. In this document, HtmlParser is the name given to the TriMedia HTML parser library.

Modules

The HtmlParser consists of several modules, each performs a different function in the HTML parser. These modules are:

- Core Parser—converts HTML into an internal token list.
- Layout—converts the token list into a display list that specifies size and position of each HTML token to be displayed. In order to determine the size and position of HTML tokens, it must know the size of the screen, the sizes of text strings according to available fonts, and the sizes of any images or widgets. That information is retrieved from the object manager.
- Navigation—builds a hotspot list and name list for navigation within the display device. Hotspot list contains hyperlinks in an HTML that jump to another page whereas name list contains hyperlinks for positions within the page.

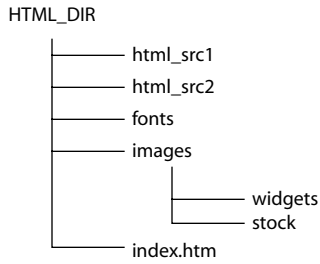
1. At present, the HTML Parser is not entirely compliant with the HTML 3.2 standard. See the section HTML Tags Supported. The HTML 3.2 standard will be fully supported in a future release.

Header Files

The header file for HtmlParser is tsaHtmlParser.h. However, there are data types shared between the HtmlParser library and the HtmlRender library. These shared data types are declared in tsaHtml.h and described in this chapter and are not duplicated in the HTML Renderer API chapter. You can distinguish these data types from the data types found in tsaHtmlParser since their prefix is tsaHtml instead of tsaHtmlParser.

Resource Files in the Database

Besides the HTML pages, the TriMedia fonts (TM font) and some widget images are important to the HTML parser. Both the fonts and widget images are always put into the particular directories where the HTML parser gets the resources. Suppose HTML_DIR is the root directory where all the resources are put into it. The figure below shows a possible directory structure in the database. The following three sections discuss how and what to put into this directory. Note that the fonts, images, widgets and stock subdirectories are required by the library whereas the html_src1 and html_src2 are provided by users. For the information about TM fonts, see Chapter 2, *2D Graphics API*. To learn how to build a database, see Chapter 7, *Object Manager (OM) API* for details.



HTML Pages

These are the HTML files. The file extension can be either 'htm' or 'html'. These files can be put under HTML_DIR or its subdirectories. The images associated with those HTML files can also be put in the same directory (for example, html_src1).

TM Fonts

For each TM font, there are two associated font files: XXX.mtr (font characteristic) and XXX.bit (font bitmap). Currently, HTML parser supports TM font only. By default, the font files are put under the subdirectory of HTML_DIR/fonts. Since the HTML parser needs the font sizes (absolute font size) ranging from 1 to 7, TM font of sizes 12, 14, 16, 18, 20, 22 and 24 are used respectively. In addition to the different font sizes, different font styles are also needed.

The followings are the font requirements:

- Default font uses plain style and size of 14 (plain14.mtr and plain14.bit).
- Regular texts need plain style fonts.
- ADDRESS, I tags need italic style fonts (e.g. italic14.mtr and italic14.bit).
- H1, H2,..., H6, B and STRONG tags need bold style fonts (e.g. bold14.mtr and bold14.bit).
- PRE and TT tags need fixed style fonts (e.g. fixed14.mtr and fixed14.bit).

You are advised to put these fonts in the directory if the tags described above are used, otherwise an error of **OM_ERR_OBJECT_NOT_IN_DATABASE** is returned from 'get object' function when **tsaHtmlParserInsatnceSetup** is called.

Widget Images

Widget images are needed when INPUT, SELECT, TEXTAREA and HSLIDER fields are used within FROM elements. INPUT can be used for a variety of from fields including single line text fields, password fields, checkboxes, buttons, radio buttons, submit and reset buttons and image buttons. SELECT elements are used for single or multiple choice menus. TEXTAREA elements are used to define multi-line text fields. The HSLIDER elements are used to define a horizontal slider such as volume control slider (will be discussed in the following section). Note that HSLIDER is TriMedia HTML parser extension which is not supported by the HTML 3.2 Reference Specification. By default, all the widget images are put under a directory of HTML_DIR/images/widgets. There are a total of 13 widget images in GIF file format. The associated file(s) for each widget is/are:

- Checkbox

  cb_off.gif—box without check

  cb_on.gif—box with check

- Horizontal Slider

  hs_l.gif—the left part of the horizontal slider

  hs_m.gif—the middle part of the horizontal slider

  hs_r.gif—the right part of the horizontal slider

  hs_sl.gif—the slider

- Radio Button

  rb_off.gif—'off' radio button

  rb_on.gif—'on' radio button

- Select Menu

  sel_dn.gif—the arrow pointing downward

  sel_lft.gif—the arrow pointing to the left

sel_rgt.gif—the arrow pointing to the right

sel_up.gif—the arrow pointing upward

- Text Cursor

  tx_caret.gif—the text box cursor

It is possible to have user designed widget images, but the file names should not be changed.

Other Image Files

Besides the widget images, there are some images for the list (Unordered Lists, UL) item. The filenames of the bullet image on UL and LI elements are:

- licircle.gif—circle bullet.
- lidisc.gif—disc bullet.
- lisquare.gif—square bullet.

Moreover, if an image cannot be found from the database, an image (broken.gif) will be used instead.

The default path to put these images is HTML_DIR/images/stock.

TriMedia Extensions to the HTML

The syntax of the two TriMedia extensions, which are not supported in the HTML 3.2 standard, to the HTML are presented here.

Button in INPUT tag

In INPUT tag, in addition to the input type of text fields, radio buttons, check boxes, etc., a input type of button is added which is used as a general purpose button such as push button and toggle button. The usage of this type attribute is the same as the others defined in the HTML 3.2 standard. The command is:

```
<INPUT type=button name=toggle value="toggle">
```

Horizontal Slider

This horizontal slider is a device that could be used as, for example, a volume control slider. The start tag of the horizontal slider is <HSLIDER> and there is no associated end tag. For example, to put a horizontal slider in a HTML page. The command is this:

```
<HSLIDER name="volume_bar" pixwidth=100 nopos=10 curpos=5>
```

There are four attributes for the HSLIDER tag. They are:

- name is the name that is assigned to the horizontal slider.
- pixwidth is the width of the horizontal slider in units of pixel.

- nopos is the number of positions of the slider.
- curpos is the current position of the slider.

Since Horizontal slider is designed as a form field, HSLIDER should be used within FORM elements. For example, the HTML code segment could be

```
...
<FORM>
   <HSLIDER name="a_slider" pixwidth=100 nopos=20 curpos=1>
   <P>
   <HSLIDER name="b_slider" pixwidth=100 nopos=10 curpos=8>
   <P>
   ...
</FORM>
...
```

How to Use the HTML Parser and HTML Renderer Libraries

This section describes the procedure to use the HTML parser and the HTML renderer APIs. Since both the HTML parser and the HTML renderer work together to perform the parsing, rendering, and navigating features, it is good to discuss these two libraries in one section. For information about the TriMedia HTML renderer API, please see Chapter 6, *HTML Renderer (HtmlRender) API*. This is the procedure:

1. Set up the instances of the 2D Graphics, Window Manager, Object Manager and Widget libraries. These instances are required for the HTML parser instance and/or the HTML renderer instance.
2. Create an instance of the HTML parser/renderer library by calling tsaHtmlParserOpen/tsaHtmlRenderOpen.
3. Call tsaHtmlParserGetInstanceSetup/tsaHtmlRenderGetInstanceSetup to get the instance setup structure.
4. Fill in any non-default setup values to the parser/renderer instance.
5. Call tsaHtmlParserSetupInstance/tsaHtmlRenderSetupInstance to complete the instance setup.

After obtaining a valid HTML parser/renderer library instance, parse HTML pages and navigate hotspot links by calling the parser and renderer APIs. For example,

1. Call tsaHtmlParserLoadUrl/tsaHtmlParserLoadHtml to get a HTML page specified by a particular URL or a HTML buffer generated by the application. Here the page is parsed and the associated render, hotspot and named lists are generated. A pointer to the parser frame state (of type tsaHtmlParserFrameState_t) data structure is returned.
2. The data structure of the parser frame state is used for the HTML renderer. A render frame state is created and returned by calling tsaHtmlRenderFrameStateCreate.
3. Call tsaHtmlRenderRenderFrame to render the HTML page. By default, the first hotspot, if any, on the page is highlighted.

Moreover, the HTML renderer also has several navigation and 'get information' functions. These functions can be called if necessary.

HTML Renderer Navigation Functions

- tsaHtmlRenderHotspot
- tsaHtmlRenderFollowNamedLink
- tsaHtmlRenderScrollScreen

HTML Renderer 'Get Information' Functions

- tsaHtmlRenderGetFrameId
- tsaHtmlRenderGetNumHotspots
- tsaHtmlRenderGetHotspot
- tsaHtmlRenderGetCurrentHotspot

To close the HTML parser and the HTML renderer instances:

1. Call **tsaHtmlRenderFrameStateDestroy** to free the memory allocated for the HTML renderer and a pointer to the parser state is returned which is used in the step 2.
2. Call **tsaHtmlParserUnload** to free the memory allocated for the parser frame state obtained from step 1.
3. Call **tsaHtmlParserClose**/**tsaHtmlRenderClose** to close the instance.

Example (exHtml) Overview

exHtml demonstrates the TriMedia HtmlParser/HtmlRender components, which uses the TriMedia WM, 2D, Widgets, and OM components. It is designed as a simple web browser, providing features such as back/forward page navigation, hotspot navigation, and up/down scrolling. It also demonstrates the use of 'widgets', an HTML extension specific to the TriMedia HtmlParser/HtmlRender libraries.

The example first sets up the WM/2D/Widgets/OM and HtmlParser/HtmlRender components with the screen size, database information, hotspot rendering preference, widget initialization and values to be loaded from the application, and callback functions for packet creation/destruction. Specific memory functions can be passed to the HtmlParser and HtmlRender. By default, it uses malloc, realloc, and free. exHtml parses each HTML page, while keeping a history to be used in back/forward page navigation. It then renders each frame in the page and processes hotspot navigation and widget control activation commands.

The example starts with displaying an HTML page which is generated by the application. The example code, a HTML generator in exHtml_ui.c, shows how to generate HTML on the fly. The index page of the demo is shown after pressing Enter key. There are three

hotspots. The first hotspot is an image (TriMedia Digital TV). By selecting this hotspot, a table of contents is shown which has nine hotspots. Each hotspot in the page demonstrates different HTML tags supported by the HtmlParser, such as inline image, table, horizontal slider, etc.

Run-time user interactions are printed as the program begins. They are also printed in instructions.htm, which is acessible from index.htm. Each command must be followed by a return when using TriMedia tmman for PC-TriMedia communication. Please note that this is tmman specific, and is not part of the HtmlParser library.

Wrapper Function: myGetObject

In exHtml, the Object Manager library is used to get the objects from its database. In some cases, if the required object cannot be not found from the database, object manager returns an error code of **OM_ERR_OBJECT_NOT_FOUND** to the application. However, such scalar error message does not give enough information of which object (and its URL) was not found from the database. In order to resolve this problem, a wrapper function, myGetObject, is used instead of calling **tsaOMGetObject** directly. myGetObject actually calls **tsaOMGetObject** and prints the error code with the associated URL if error occurs.

The wrapper function can be found in the example/exHtml/exHtml_init.c.

HTML Data Structures

This section presents the shared HTML data structures.

Name	Page
tsaHtmlFont_t	142
tsaHtmlWidgetStateGeneric_t	142
tsaHtmlWidgetStateTextline_t	143
tsaHtmlWidgetStatePassword_t	144
tsaHtmlWidgetStateRadio_t	145
tsaHtmlWidgetStateCheckbox_t	146
tsaHtmlWidgetStateButton_t	147
tsaHtmlWidgetStateSubmit_t	148
tsaHtmlWidgetStateReset_t	149
tsaHtmlWidgetStateImage_t	150
tsaHtmlWidgetStateFile_t	151
tsaHtmlWidgetStateHidden_t	152
tsaHtmlWidgetStateSelect_t	153
tsaHtmlWidgetStateTextarea_t	155
tsaHtmlWidgetStateSlider_t	156

tsaHtmlFont_t

```
typedef struct {
   ptsa2DFont_t         font;
   Int                  size;
   Int                  color;
   tsaHtmlFontStyle_t   style;
} tsaHtmlFont_t, *ptsaHtmlFont_t;
```

Fields

`font`	The TriMedia 2D graphics library font.
`size`	The size of the font (HTML level 1–7).
`color`	The color of the font (an HTML color).
`style`	The style (e.g., bold) of the font.

Description

This data type specifies the font, color, and style to be used in rendering the text.

tsaHtmlWidgetStateGeneric_t

```
typedef struct {
   Int                  id;
   Char                *name;
   ptsa2DFont_t         font;
} tsaHtmlWidgetStateGeneric_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the widget.
`font`	Pointer to the TM font used.

Description

This is the header of each widget state data structure.

tsaHtmlWidgetStateTextline_t

```
typedef struct {
   Int            id;
   Char          *name;
   ptsa2DFont_t   font;
   Char          *text;
   Int            size;
   Int            maxLength;
   Int            firstChar;
   Int            cursorPos;
   Bool           showCursor;
   Char          *cursor_image_url;
} tsaHtmlWidgetStateTextline_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the textline widget.
`font`	Pointer to the TM font used.
`text`	Pointer to the text (the value attribute of the input field).
`size`	Visible size of the text box (the size attribute of the input field).
`maxLength`	Maximum number of characters permitted to be entered.
`firstChar`	Offset of the first character that is visible in the text box.
`cursorPos`	Cursor position.
`showCursor`	True, shows the cursor.
`cursor_image_url`	URL of the cursor image.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to text.

For example, the HTML would be

```
<INPUT type=text size=40 name=user value="your name">
```

tsaHtmlWidgetStatePassword_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *text;
   Int             size;
   Int             maxLength;
   Int             firstChar;
   Int             cursorPos;
   Bool            showCursor;
   Char           *cursor_image_url;
} tsaHtmlWidgetStatePassword_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the password widget.
`font`	Pointer to the TM font used.
`text`	Pointer to a text (the value attribute of the input field).
`size`	Visible size of the password box (the size attribute of the input field).
`maxLength`	Maximum number of characters permitted to be entered.
`firstChar`	Offset of the first character that is visible in the text box.
`cursorPos`	Cursor position.
`showCursor`	True, shows the cursor.
`cursor_image_url`	URL of the cursor image.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to password.

For example, the HTML would be

```
<INPUT type=password size=12 name=pw>
```


tsaHtmlWidgetStateRadio_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *value;
   Bool            checked;
   Char           *on_image_url;
   Char           *off_image_url;
} tsaHtmlWidgetStateRadio_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the radio button.
`font`	Pointer to the TM font used.
`value`	A text string from the value attribute of the input field.
`checked`	Checked state of the radio button.
`on_image_url`	URL of the 'checked' image.
`off_image_url`	URL of the 'unchecked' image.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to **radio**.

For example, the HTML would be

```
<INPUT type=radio name=age value="0-12">
<INPUT type=radio name=age value="13-17">
<INPUT type=radio name=age value="18-25">
<INPUT type=radio name=age value="26-35" checked>
<INPUT type=radio name=age value="36-">
```

tsaHtmlWidgetStateCheckbox_t

```
typedef struct {
   Int            id;
   Char          *name;
   ptsa2DFont_t   font;
   Char          *value;
   Bool           checked;
   Char          *on_image_url;
   Char          *off_image_url;
} tsaHtmlWidgetStateCheckbox_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the checkbox widget.
`font`	Pointer to the TM font used.
`value`	A text string from the value attribute of the input field.
`checked`	Checked state of the checkbox.
`on_image_url`	URL of the 'checked' image.
`off_image_url`	URL of the 'unchecked' image.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to checkbox.

For example, the HTML would be

```
<INPUT type=checkbox checked name=uscitizen value=yes>
```


tsaHtmlWidgetStateButton_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *value;
} tsaHtmlWidgetStateButton_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the button widget.
`font`	Pointer to the TM font used.
`value`	A text string from the value attribute of the input field.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to button.

For example, the HTML would be

```
<INPUT type=button name=toggle value="toggle">
```

Note that this is a TriMedia extension only which does not support in HTML 3.2 standard.

tsaHtmlWidgetStateSubmit_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *value;
} tsaHtmlWidgetStateSubmit_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the submit widget.
`font`	Pointer to the TM font used.
`value`	A text string from the value attribute of the input field.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to submit.

For example, the HTML would be

```
<INPUT type=submit value="Party on...">
```


tsaHtmlWidgetStateReset_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *value;
} tsaHtmlWidgetStateReset_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the reset widget.
`font`	Pointer to the TM font used.
`value`	A text string from the value attribute of the input field.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to reset.

For example, the HTML would be

```
<INPUT type=submit value="Start over...">
```

tsaHtmlWidgetStateImage_t

```
typedef struct {
   Int                  id;
   Char                *name;
   ptsa2DFont_t         font;
   Char                *src;
   tsaHtmlImageAlign_t  align;
   Int                  x;
   Int                  y;
} tsaHtmlWidgetStateImage_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the image widget.
`font`	Pointer to the TM font used.
`src`	URL of the image.
`align`	Alignment of the image (the align attribute of the input field).
`x`	x position of the click.
`y`	y position of the click.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to image.

For example, the HTML would be

```
<INPUT type=image name=point align=middle
src="file:\\\images\stock\map.gif">
```

Note the x and y values of the location clicked are not supported in the current release.

tsaHtmlWidgetStateFile_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *text;
   Int             size;
   Int             maxLength;
   Char           *accept;
   Int             firstChar;
   Int             cursorPos;
   Bool            showCursor;
   Char           *cursor_image_url;
   Char           *browser_image_url;
} tsaHtmlWidgetStateFile_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the file widget.
`font`	Pointer to the TM font used.
`text`	Pointer to the text (the value attribute of the input field).
`size`	Visible size of the text box (the size attribute of the input field).
`maxLength`	Maximum number of characters permitted to be entered.
`accept`	Pointer to the text which is specified by the accept attribute of the input field.
`firstChar`	Offset of the first character that is visible in the text box.
`cursorPos`	Cursor position.
`showCursor`	True, shows the cursor.
`cursor_image_url`	URL of the cursor image.
`browser_image_url`	URL of the file browser image button.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to file.

For example, the HTML would be

```
<INPUT type=file name=photo size=20 accept="image/*">
```

tsaHtmlWidgetStateHidden_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char           *value;
} tsaHtmlWidgetStateHidden_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the hidden widget.
`font`	Pointer to the TM font used.
`value`	A text string from the value attribute of the input field.

Description

This data structure is used when using INPUT tag with the type attribute of the input field set to hidden.

For example, the HTML would be

```
<INPUT type=hidden name=customerid value="c2415-345-8563">
```


tsaHtmlWidgetStateSelect_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Int             size;
   Bool            multiple;
   Int             numOptions;
   Char          **optionText;
   Char          **optionValue;
   Bool           *optionSelected;
   Int             maxLength;
   Int             topPos;
   Int             cursorPos;
   Char           *uparrow_image_url;
   Char           *dnarrow_image_url;
   Char           *larrow_image_url;
   Char           *rarrow_image_url;
} tsaHtmlWidgetStateSelect_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the select widget.
`font`	Pointer to the TM font used.
`size`	Number of visible options.
`multiple`	True, multiple selections.
`numOptions`	Total number of options found from HTML.
`optionText`	An array of text for each option.
`optionValue`	An array of value for each option.
`optionSelected`	An array to indicate which option is selected or not.
`maxLength`	Maximum number of characters allowed for each option.
`topPos`	Offset to the first option to be displayed first.
`cursorPos`	Cursor position.
`uparrow_image_url`	URL of the up arrow image.
`dnarrow_image_url`	URL of the down arrow image.
`larrow_image_url`	URL of the left arrow image.
`rarrow_image_url`	URL of the right arrow image.

Description

This data structure is used when SELECT tag is used.

For example, the HTML would be

```
<SELECT name="flavor">
<OPTION value=a>Vanilla
<OPTION value=b>Strawberry
<OPTION value=c>Rum and Raisin
<OPTION value=d>Peach and Orange
</SELECT>
```


tsaHtmlWidgetStateTextarea_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Char          **text;
   Int             rows;
   Int             cols;
   Int             rowOffset;
   Int             colOffset;
   Int             cursorRow;
   Bool            cursorCol;
   Int             showCursor;
   Char           *cursor_image_url;
} tsaHtmlWidgetStateTextarea_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the textarea widget.
`font`	Pointer to the TM font used.
`text`	An array of text bracketed by TEXTAREA tag.
`rows`	Number of visible text lines (the rows attribute).
`cols`	The visible width in average character widths (the cols attribute).
`rowOffset`	Top visible row.
`colOffset`	First visible character for each row.
`cursorRow`	Row position of the cursor.
`cursorCol`	Cursor position within the row.
`showCursor`	True, shows the cursor.
`cursor_image_url`	URL of the cursor image.

Description

This data structure is used when TEXTAREA tag is used.

For example, the HTML would be

```
<TEXTAREA name=address rows=4 cols=40>
Your address here...
</TEXTAREA>
```

tsaHtmlWidgetStateSlider_t

```
typedef struct {
   Int             id;
   Char           *name;
   ptsa2DFont_t    font;
   Int             pixWidth;
   Int             maxPositions;
   Int             curPosition;
   Char           *right_image_url;
   Char           *left_image_url;
   Char           *mid_image_url;
   Char           *tab_image_url;
} tsaHtmlWidgetStateSilder_t;
```

Fields

`id`	The widget ID.
`name`	The property name of the slider widget.
`font`	Pointer to the TM font used.
`pixWidth`	Slider width in pixel (the pixwidth attribute).
`maxPositions`	Total number of position (the nopos attribute).
`curPosition`	Current position of the slider (the curpos attribute).
`right_image_url`	URL of the right part of the slider.
`left_image_url`	URL of the left part of the slider.
`mid_image_url`	URL of the middle part of the slider.
`tab_image_url`	URL of the slider tab of the slider.

Description

This data structure is used when SLIDER tag is used.

For example, the HTML would be

```
<HSLIDER name="volume_bar" pixwidth=100 nopos=10 curpos=5>
```

Note that this is a TriMedia extension only which does not support in HTML 3.2 standard.

HTML Enumerated Types

This section presents the shared HTML enumerated types.

Name	Page
tsaHtmlHotspotType_t	158
tsaHtmlFontStyle_t	159
tsaHtmlImageAlign_t	159

tsaHtmlHotspotType_t

```
typedef enum {
   tsaHtmlHotspotTypeLinkText       = 0x101,
   tsaHtmlHotspotTypeLinkClient     = 0x102,
   tsaHtmlHotspotTypeLinkServer     = 0x103,
   tsaHtmlHotspotTypeLinkError      = 0x104,
   tsaHtmlHotspotTypeWidgetTextline = 0x201,
   tsaHtmlHotspotTypeWidgetPassword = 0x202,
   tsaHtmlHotspotTypeWidgetRadio    = 0x203,
   tsaHtmlHotspotTypeWidgetCheckbox = 0x204,
   tsaHtmlHotspotTypeWidgetButton   = 0x205,
   tsaHtmlHotspotTypeWidgetSubmit   = 0x206,
   tsaHtmlHotspotTypeWidgetReset    = 0x207,
   tsaHtmlHotspotTypeWidgetFile     = 0x208,
   tsaHtmlHotspotTypeWidgetImage    = 0x209,
   tsaHtmlHotspotTypeWidgetHidden   = 0x210,
   tsaHtmlHotspotTypeWidgetSelect   = 0x211,
   tsaHtmlHotspotTypeWidgetTextarea = 0x212,
   tsaHtmlHotspotTypeWidgetSlider   = 0x213,
} tsaHtmlHotspotType_t ;
```

Description

This enumerates all possible types of hotspots supported by TriMedia HTML.

tsaHtmlFontStyle_t

```
typedef enum {
   tsaHtmlFontStyleNormal  =  0,
   tsaHtmlFontStyleBold    =  1,
   tsaHtmlFontStyleItalic  =  2,
   tsaHtmlFontStyleFixed   =  4,    // fixed pitch
   tsaHtmlFontStyleStrike  =  8,    // strike-through
   tsaHtmlFontStyleUnder   = 16     // underlined
} tsaHtmlFontStyle_t;
```

Description

This enumerates all possible HTML font styles.

tsaHtmlImageAlign_t

```
typedef enum {
   tsaHtmlAlignImageLeft,
   tsaHtmlAlignImageRight,

   tsaHtmlAlignImageTop,
   tsaHtmlAlignImageMiddle,
   tsaHtmlAlignImageBottom,

   tsaHtmlAlignImageTexttop,
   tsaHtmlAlignImageAbsmiddle,
   tsaHtmlAlignImageBaseline,
   tsaHtmlAlignImageAbsbottom
} tsaHtmlImageAlign_t;
```

Description

This enumerates various image alignment options.

HTML API Data Structures

This section presents the HtmlParser API data structures.

Name	Page
tsaHtmlParserCapabilities_t	161
tsaHtmlParserInstanceSetup_t	162
tsaHtmlParserFrameState_t	163
tsaHtmlParserSetupFlags_t	164

tsaHtmlParserCapabilities_t

```
typedef struct {
   ptmolDefaultCapabilities_t   defaultCapabilities;
} tsaHtmlParserCapabilities_t, *ptsaHtmlParserCapabilities_t;
```

Fields

`defaultCapabilities`	Pointer to a default capabilities structure (see tsa.h).

Description

The structure describes the capabilities of HtmlParser. The parser does not have any other capabilities data other than those in **defaultCapabilities**.

tsaHtmlParserInstanceSetup_t

```
typedef struct {
   Int                          tsa2DInst;
   Int                          tsaOMInst;
   tsaHtmlParserSetupFlags_t    flags;
   Int                          ScreenWidth;
   Int                          ScreenHeight;
   void                      *(*MallocFn)(size_t size);
   void                      *(*ReallocFn)(void *ptr, size_t size);
   void                       (*FreeFn)(void *ptr);
   tmLibappErr_t              (*GetObjectFn)(
                                Int instance,
                                Char *Url,
                                tsaOMType_t type,
                                Pointer *pObject );
} tsaHtmlParserInstanceSetup_t, *ptsaHtmlParserInstanceSetup_t;
```

Fields

`tsa2DInst`	The 2D instance previously opened and set up by the application. HtmlParser uses this to obtain text rendering information.
`tsaOMInst`	The OM (object manager) instance previously opened and set up by the application. HtmlParser uses this as first argument to the **GetObjectFn** callback.
`ScreenWidth`	The display's width. The default value is 720.
`ScreenHeight`	The display's height. The default value is 480.
`MallocFn`	Callback memory allocation function.
`ReallocFn`	Callback reallocation function.
`FreeFn`	Callback free function.
`GetObjectFn`	Callback 'get object' function used by HtmlParser to obtain database objects. Typically, you would use the OM function **tsaOMGetObject**.

Description

The structure holds initial information, from the application, to set up the HtmlParser instance. Not all members of this structure need be set before setting up the HtmlParser instance. The HtmlParser will use default values for each member not set by the application before instance setup.

tsaHtmlParserFrameState_t

```
typedef struct ParserFrameState_t {
   Int                          frameId;
   Char                        *url;
   Int                          x;
   Int                          y;
   Int                          width;
   Int                          height;
   void                        *background;
   Int                          textcolor;
   Int                          linkcolor;
   Int                          vlinkColor;
   Int                          alinkColor;
   Int                          endline;
   char                        *renderList;
   char                        *widgetStates;
   int                          numHotspots;
   HOTSPOT                     *hotspotList;
   int                          numAnchors;
   NAMEDLINK                   *anchorList;
   int                          numSubFrames;
   struct ParserFrameState_t   *subFrames;
} tsaHtmlParserFrameState_t, *ptsaHtmlParserFrameState_t;
```

Fields

`frameId`	Identifier for this frame.
`url`	URL name of this page.
`x`	X offset of the frame on screen.
`y`	Y offset of the frame on screen.
`width`	Width of the frame.
`height`	Height of the frame.

The remaining fields are used by the HtmlRender library. You should not use their values directly.

Description

This structure passes information from the HtmlParser library to the HtmlRender library. Other than the fields described above, you should not inspect or store values in the remaining fields.

tsaHtmlParserSetupFlags_t

```
typedef enum {
   tsaHtmlParserFlagNone = 0x00000000,
} tsaHtmlParserSetupFlags_t;
```

Description

This enumerates flags in the instance setup structure. Currently there are no flags.

HTML API Functions

This section presents the HTML Parser API library functions.

Name	Page
tsaHtmlParserGetCapabilities	166
tsaHtmlParserOpen	167
tsaHtmlParserGetInstanceSetup	168
tsaHtmlParserInstanceSetup	169
tsaHtmlParserClose	170
tsaHtmlParserLoadUrl	171
tsaHtmlParserLoadHtml	172
tsaHtmlParserUnload	173

tsaHtmlParserGetCapabilities

```
tmLibappErr_t tsaHtmlParserGetCapabilities(
   tsaHtmlParserCapabilities_t   **cap
);
```

Parameters

`cap`	Pointer to a variable in which to return a pointer to the parser capabilities structure.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Returns a pointer to the parser capabilities.

tsaHtmlParserOpen

```
tmLibappErr_t tsaHtmlParserOpen(
   Int   *instance
);
```

Parameters

`instance`	Pointer (returned) to the instance.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NO_INSTANCE_AVAILABLE`	No instance is available. Because HtmlParser is not yet re-entrant, only one instance can run at any given time.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory was available to allocate instance variables.

Description

Allocates an instance of the HTML parser. Initializes the setup structure to default values.

tsaHtmlParserGetInstanceSetup

```
tmLibappErr_t tsaHtmlParserGetInstanceSetup(
   Int                              instance,
   ptsaHtmlParserInstanceSetup_t *setup
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlParserOpen**.
`setup`	Pointer to a variable in which to return a pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.

Description

Returns the instance setup structure.

tsaHtmlParserInstanceSetup

```
tmLibappErr_t tsaHtmlParserInstanceSetup(
   Int                              instance,
   tsaHtmlParserInstanceSetup_t  *setup
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlParserOpen**.
`setup`	Pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`HTMLPARSER_ERR_NULL_WININST`	WMInst in setup structure not filled with valid WM instance id.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory.
`HTMLPARSER_ERR_NULL_GET_OBJECT_FUNC`	No callback 'get object' function is available.

Description

Sets up the instance of HtmlParser according to the setup structure in the opened instance. Loads a default font, a TM plain style font of the size of 14, from the object manager database. Note that the database must have this font, plain14.mtr and plain14.bit, otherwise an error, **OM_ERR_OBJECT_NOT_IN_DATABASE** is returned when 'get object' function is trying to load this font. (See *Resource Files in the Database* on page 135, Resource Files in the Database and Chapter 7, *Object Manager (OM) API* for details.)

tsaHtmlParserClose

```
tmLibappErr_t tsaHtmlParserClose(
   Int   instance
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlParserOpen**.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.

Description

Deallocates the instance previously opened by **tsaHtmlParserOpen**. Closes the parser and widget modules. Free all memory associated with the instance.

tsaHtmlParserLoadUrl

```
tmLibappErr_t tsaHtmlParserLoadUrl(
   Int                          instance,
   char                        *Url,
   ptsaHtmlParserFrameState_t  *frameStates
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlParserOpen**.
`Url`	The name of the HTML page to be loaded.
`frameStates`	A pointer to a variable in which to return a pointer to a parser frame state structure.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	`Not enough memory.`
`HTMLPARSER_ERR_NULL_GET_OBJECT_FUNC`	No callback 'get object' function is available.

Description

Loads the HTML page specified by the URL and returns a pointer to the parser state data structure which contains the information from the HtmlParser library to the Html-Render library.

tsaHtmlParserLoadHtml

```
tmLibappErr_t tsaHtmlParserLoadHtml(
   Int                           instance,
   char                         *Url,
   ptsaHtmlParserFrameState_t   *frameStates,
   Char                         *data,
   Int                           size
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlParserOpen**.
`Url`	The name of the HTML page, identifying the page.
`frameStates`	A pointer to a variable in which to return a parser frame state structure.
`data`	A pointer to the buffer containing the HTML page description.
`size`	The size of the buffer containing the HTML page description.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory.

Description

Loads the HTML page in the buffer given as described by data and size and returns a pointer to the parser state data structure which contains the information from the HtmlParser library to the HtmlRender library.

tsaHtmlParserUnload

```
tmLibappErr_t tsaHtmlParserUnload(
   Int                          instance,
   ptsaHtmlParserFrameState_t   frameStates
);
```

Parameters

`instance`	The instance.
`frameStates`	A pointer to the parser frame state structure.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.

Description

Unloads the parser state which is returned from **tsaHtmlRenderFrameStateDestroy**.

HTML Tags Supported

The HtmlParser module currently supports most of HTML 3.2 with a few minor exceptions. The supported HTML tags and attributes are listed in Table 2. Those tags that are not yet supported are listed in Table 3. Attributes marked with (N) are Netscape extensions to the standard. It should be noted that the parser module will skip over any tag that is not yet supported so pages containing these tags can be safely parsed without generating any errors.

Table 2 Supported HTML Tags

tag	attributes	parser support	layout support
A	href name rel rev title	yes yes yes yes yes	yes
ADDRESS	none	yes	yes
AREA	alt href nohref shape coords		yes
B	none	yes	yes
BASE	href target (N)	yes yes	yes
BLOCKQUOTE	none	yes	yes
BODY	text link alink vlink bgcolor background	yes yes yes yes yes yes	yes
BR	clear	yes	yes
CAPTION	align	yes	no
CENTER	none	yes	yes
DD	none	yes	yes
DL	compact	yes	yes
DT	none	yes	yes
FONT	size color	yes yes	yes

Table 2 Supported HTML Tags

FORM	action method enctype	yes yes yes	yes
FRAME	src name noresize marginwidth marginheight	yes yes yes yes yes	yes
FRAMESET	rows cols border bordercolor framespacing	yes yes yes yes yes	yes
H1	align	yes	yes
H2	align	yes	yes
H3	align	yes	yes
H4	align	yes	yes
H5	align	yes	yes
H6	align	yes	yes
HEAD	none	yes	yes
HR	align noshade size width	yes	yes
HSLIDER[A]	name pixWidth nopos curpos	yes yes yes yes	 yes yes
HTML	none	yes	yes
I	none	yes	yes
IMG	alt src align width height border hspace vspace ismap usemap	yes yes yes yes yes yes yes yes yes yes	yes yes yes yes yes no no no yes yes

Table 2 Supported HTML Tags

INPUT	type name value align checked src size maxlength	yes yes yes yes yes yes yes yes	yes
LI	type value	yes	yes
MAP	name	yes	yes
NOFRAMES	none	yes	yes
OL	type start compact	yes yes yes	yes yes no
OPTION	value selected	yes yes	yes
P	align	yes	yes
PRE	width	yes	yes
SELECT	name size multiple	yes yes yes	yes
STRONG	none	yes	yes
TABLE	align width border bgcolor (N) cellspacing cellpadding	yes yes yes yes yes yes	yes
TD, TH	align valign width height rowspan colspan bgcolor (N) nowrap	yes yes yes yes yes yes yes yes	yes yes yes yes yes yes yes no
TEXTAREA	name cols rows	yes yes yes	yes
TITLE	none	yes	no

Table 2 Supported HTML Tags

TR	align valign bgcolor (N)	yes yes yes	yes
TT	none	yes	yes
UL	type compact	yes yes	yes no

A. HSLIDER is not supported by the HTML 3.2 Reference Specification. It is the TriMedia HTML parser extension.

Table 3 Unsupported HTML Tags

tag	attribute	parser support	layout support
BIG	none	no	no
CITE	none	no	no
CODE	none	no	no
DFN	none	no	no
DIR	none	no	no
DIV	none	no	no
EM	none	no	no
ISINDEX	prompt	no	no
KBD	none	no	no
LINK	none	no	no
LISTING	*OBSOLETE*	no	no
MENU	none	no	no
META	none	no	no
PARAM	none	no	no
PLAINTEXT	*OBSOLETE*	no	no
SAMP	none	no	no
SCRIPT	none	no	no
SMALL	none	no	no
STRIKE	none	no	no
STYLE	none	no	no
SUB	none	no	no
SUP	none	no	no

Table 3 Unsupported HTML Tags

U	none	no	no
VAR	none	no	no
XMP	*OBSOLETE*	no	no

Chapter 6

HTML Renderer (HtmlRender) API

Topic	Page
Overview	134
HTML Renderer API Data Structures	181
HTML Renderer API Functions	188

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

Overview

The TriMedia HTML renderer library processes the output of the HTML parser and calls the TriMedia 2D graphics library and widget library to render text and images into video buffers, which are then put on the screen by the TriMedia window manager. A window manager instance is passed to the HtmlRender during setup.

HtmlRender supports navigation and the rendering of hotspots. The HtmlRender library also supports vertical scrolling of the HTML page when the page height is greater than the displayed window height.

The example for the HtmlRender library is exHtml which is discussed in the HtmlParser API document.

Modules

The HtmlRender consists of several modules each performs a different function in the HTML renderer. These modules are:

- HTML Page Rendering—renders the parsed HTML page to the HtmlRender frame.
- Navigation—navigates the hotspots and scrolls the rendered frame up/down if the HTML page height is greater than the displayed window height.
- Get Information—gets the information about the renderer frame state and hotspots.

Header Files

The header file for the HtmlRender is tsaHtmlRender.h. However, there are data types that are shared between the HTML parser and the HTML renderer. These data types are declared in tsaHtml.h and are described in the HtmlParser chapter and not duplicated in this chapter. Please see the TriMedia HtmlParser API chapter for descriptions of the shared data types.

The TriMedia HTML Parser (HtmlParser)

Since the HtmlRender library uses the results, the parser state information, from the HtmlParser library to perform the rendering and navigating features, users are advised to read the HtmlParser API document also. Moreover, a section in the HtmlParser API document is provided to describe how to use both the HTML parser and renderer libraries.

HTML Renderer API Data Structures

This section presents the HtmlRender API data structures.

Name	Page
tsaHtmlRenderCapabilities_t	182
tsaHtmlRenderInstanceSetup_t	183
tsaHtmlRenderWidgetState_t	185
tsaHtmlRenderSetupFlags_t	186
tsaHtmlRenderHotspotDir_t	186
tsaHtmlRenderScrollDir_t	187

tsaHtmlRenderCapabilities_t

```
typedef struct {
   ptmolDefaultCapabilities_t   defaultCapabilities;
} tsaHtmlRenderCapabilities_t, *ptsaHtmlRenderCapabilities_t;
```

Fields

`defaultCapabilities`	Pointer to a default capabilities structure (see tsa.h).

Description

This structure describes the capabilities of HtmlRender. HtmlRender does not have any capabilities other than those in **defaultCapabilities**.

tsaHtmlRenderInstanceSetup_t

```
typedef struct {
   Int                          tsa2DInst;
   Int                          tsaWMInst;
   Int                          tsaOMInst;
   Int                          tsaWidgetInst;
   tsaHtmlRenderSetupFlags_t    flags;
   Int                          ScreenWidth;
   Int                          ScreenHeight;
   Int                          ScreenVOff;
   Int                          ScreenHOff;
   void                      *(*MallocFn)(size_t size);
   void                      *(*ReallocFn)(void *ptr, size_t size);
   void                       (*FreeFn)(void *ptr);
   ptmAvPacket_t              (*CreatePkt)(
                                Int rootWinWidth,
                                Int rootWinHeight);
   void                       (*DestroyPkt)(ptmAvPacket_t pkt);
   tmLibappErr_t              (*GetObjectFn)(
                                Int instance, Char *Url,
                                tsaOMType_t type,
                                Pointer *Object);
   ptsa2DColor_t                transparent;
   Int                          alpha;
} tsaHtmlRenderInstanceSetup_t, *ptsaHtmlRenderInstanceSetup_t;
```

Fields

`tsa2DInst`	The 2D instance previously opened and set up by the application. HtmlRender uses this to draw the HTML contents.
`tsaWMInst`	The WM (window manager) instance previously opened and set up by the application. HtmlRender uses this to manage the window (frames).
`tsaOMInst`	The OM (object manager) instance previously opened and set up by the application. HtmlRender uses this as first argument to the 'get object' callback function.
`tsaWidgetInst`	The Widget instance previously opened and setup by the application. HtmlRender uses this to draw the widgets.
`ScreenWidth`	The display's width. The default value is 720.
`ScreenHeight`	The display's height. The default value is 480.
`ScreenVOff`	The display's vertical offset. The default value is 0.
`ScreenHOff`	The display's horizontal offset. The default value is 0.

`MallocFn`	Callback memory allocation function.
`ReallocFn`	Callback reallocation function.
`FreeFn`	Callback free function.
`CreatePkt`	Callback function to create a packet with specific width, height, and YUV buffers.
`DestroyPkt`	Callback function to destroy a packet created by **CreatePkt**.
`GetObjectFn`	Callback function to obtain database objects. Typically, you would use a wrapper function, **myGetObject** which calls **tsaOMGetObject**.
`transparent`	The tsa2D color used for the color "transparent."
`alpha`	The alpha value applied to colors when the output buffer type is **vdfYUV422PlanarAlpha4**.

Description

The structure holds initial information, from the application, to set up the HtmlRender instance.

tsaHtmlRenderWidgetState_t

```
typedef struct {
   tsaWidget_t   widgetId;
   UInt32        winId;
   Boolean       displayed;
   Pointer       initState;
   Int           formId;
} tsaHtmlRenderWidgetState_t, *ptsaHtmlRenderWidgetState_t;
```

Fields

`widgetId`	The Widget library object.
`winId`	The Window Manager window ID.
`displayed`	True if widget is displayed on screen currently.
`initState`	The initial state of the widget objects.
`formId`	The ID of the form of the widget.

Description

This is the structure returned by **tsaHtmlRenderGetCurrentHotspot** or **tsaHtmlRenderGetHotspot**. Note that **initState** should be casted to a variable type specified by the hotspot type.

tsaHtmlRenderSetupFlags_t

```
typedef enum {
   tsaHtmlRenderFlagNone                = 0x00000000,
   tsaHtmlRenderFlagHotspotActionNone   = 0x00000001,
   tsaHtmlRenderFlagHotspotActionBorder = 0x00000002,
} tsaHtmlRenderSetupFlags_t;
```

Description

This enumerates flags in the instance setup.

The value **tsaHtmlRenderFlagHotspotActionBorder** causes hotspots to be highlighted with a rectangular border when activated.

tsaHtmlRenderHotspotDir_t

```
typedef enum {
   tsaHtmlRenderHotspotFirst  = 0,
   tsaHtmlRenderHotspotUp     = 1,
   tsaHtmlRenderHotspotDown   = 2,
   tsaHtmlRenderHotspotLeft   = 3,
   tsaHtmlRenderHotspotRight  = 4,
   tsaHtmlRenderHotspotInView = 5,
   tsaHtmlRenderHotspotSelect = 6,
} tsaHtmlRenderHotspotDir_t;
```

Description

This enumerates the directions of the next hotspot to activate. It is used as an argument to **tsaHtmlRenderRenderHotspot**.

The value **tsaHtmlRenderHotspotFirst** activates the first hotspot on the HTML page.

The values

```
tsaHtmlRenderHotspotLeft    tsaHtmlRenderHotspotUp
tsaHtmlRenderHotspotRight   tsaHtmlRenderHotspotDown
```

are directions in the two-dimensional list of hotspots, if one was created.

The value **tsaHtmlRenderHotspotInView** activates the first hotspot in current view, if the HTML page is larger than the display window and the current view is not the top of the HTML page.

The value **tsaHtmlRenderHotspotSelect** can be used to select any specific hotspot on the HTML page.

tsaHtmlRenderScrollDir_t

```
typedef enum {
   tsaHtmlRenderScrollUp     = 0,
   tsaHtmlRenderScrollDown   = 1,
} tsaHtmlRenderScrollDir_t;
```

Description

This enumerates the direction in which to scroll the screen. It is used as an argument to the function, **tsaHtmlRenderScrollScreen**. Because long lines are wrapped during the layout process, left and right scrolling is not necessary and not supported.

HTML Renderer API Functions

This section presents the HTML Render API library functions.

Name	Page
tsaHtmlRenderGetCapabilities	189
tsaHtmlRenderOpen	190
tsaHtmlRenderGetInstanceSetup	191
tsaHtmlRenderInstanceSetup	192
tsaHtmlRenderClose	193
tsaHtmlRenderFrameStateCreate	194
tsaHtmlRenderFrameStateDestroy	195
tsaHtmlRenderRenderFrame	196
tsaHtmlRenderRenderAllFrames	197
tsaHtmlRenderRenderHotspot	198
tsaHtmlRenderGetFrameId	199
tsaHtmlRenderGetCurrentHotspot	200
tsaHtmlRenderGetHotspot	201
tsaHtmlRenderGetNumHotspots	202
tsaHtmlRenderGetSubFrame	203
tsaHtmlRenderGetNumSubFrames	204
tsaHtmlRenderFollowNamedLink	205
tsaHtmlRenderScrollScreen	206

tsaHtmlRenderGetCapabilities

```
tmLibappErr_t tsaHtmlRenderGetCapabilities(
   tsaHtmlRenderCapabilities_t   **cap
);
```

Parameters

`cap`	Pointer to a variable in which to return a pointer to a renderer capabilities structure.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Returns a pointer to the renderer capabilities.

tsaHtmlRenderOpen

```
tmLibappErr_t tsaHtmlRenderOpen(
   Int   *instance
);
```

Parameters

`instance`	Pointer (returned) to the instance.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NO_INSTANCE_AVAILABLE`	No instance is available. Because HtmlRender is not yet re-entrant, only one instance can run at any given time.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory was available to allocate instance variables.

Description

Allocates an instance of HtmlRender. Initializes the setup structure to default values.

tsaHtmlRenderGetInstanceSetup

```
tmLibappErr_t tsaHtmlRenderGetInstanceSetup(
   Int                              instance,
   ptsaHtmlRenderInstanceSetup_t  *setup
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`setup`	Pointer to a variable in which to return a pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.

Description

Gets the instance setup structure.

tsaHtmlRenderInstanceSetup

```
tmLibappErr_t tsaHtmlRenderInstanceSetup(
   Int                              instance,
   tsaHtmlRenderInstanceSetup_t  *setup
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`setup`	Pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`HTMLRENDER_ERR_NULL_WININST`	WMInst in setup structure not filled with valid WM instance id.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory available to allocate object database.
`HTMLRENDER_ERR_NULL_DATABASE`	Data in setup structure not filled with valid database.
`HTMLRENDER_ERR_NO_2D_INSTANCE`	tsa2DInst instance in setup structure not filled with valid 2D instance id.
`HTMLRENDER_ERR_NO_WM_INSTANCE`	tsaWMInst in setup structure not filled with valid WM instance id.
`HTMLRENDER_ERR_NO_WIDGET_INSTANCE`	tsaWidgetInst in setup structure not filled with valid widget instance id.
`HTMLRENDER_ERR_NO_CREATEPKT_FUNC`	CreatePkt in setup structure not filled with valid 'create packet' function pointer.
`HTMLRENDER_ERR_NO_DESTROYPKT_FUNC`	DestroyPkt in setup structure not filled with valid 'destroy packet' function pointer.

Sets up the instance of HtmlRender according to the setup structure in the opened instance.

Description

Sets up the instance of HtmlRender according to the setup structure in the opened instance.

tsaHtmlRenderClose

```
tmLibappErr_t tsaHtmlRenderClose(
   Int   instance
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.

Description

Deallocates the instance previously open in tsaHtmlRenderOpen. Frees all memory associated with the instance.

tsaHtmlRenderFrameStateCreate

```
tmLibappErr_t tsaHtmlRenderFrameStateCreate(
   Int                          instance,
   ptsaHtmlParserFrameState_t   parserState,
   Int                         *frameState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`parserState`	The parser state, from the HTML parser.
`frameState`	Pointer to a variable in which to return the renderer frame state.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory was available.
`HTMLRENDER_ERR_INVALID_WIDGET_TYPE`	The widget type is not supported. (see the Widget API documentation for supported widget types).

Description

Creates the HtmlRender frame state, given the HTML parser state. The returned HtmlRender frame state is a parameter in many of the HtmlRender API functions.

Note that once **parserState** is passed into **tsaHtmlRenderFrameStateCreate**, it cannot be freed (calling **tsaHtmlParserUnload**) until the renderer is done (calling **tsaHtmlRenderFrameStateDestroy**).

tsaHtmlRenderFrameStateDestroy

```
tmLibappErr_t tsaHtmlRenderFrameStateDestroy(
   Int                          instance,
   Int                          frameState,
   ptsaHtmlParserFrameState_t   *parserState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.
`parserState`	The HtmlParser parser state.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.

Description

Destroys the HtmlRender frame state (deallocates memory) and returns the associated parser state to the application so that the application can then call **tsaHtmlParserUnload** to free the **parserState** data structure.

tsaHtmlRenderRenderFrame

```
tmLibappErr_t tsaHtmlRenderRenderFrame(
   Int   instance,
   Int   frameState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory was available.
`HTMLRENDER_ERR_NULL_GET_OBJECT_FUNC`	No 'get object' function is available.

Description

Renders the frame given by frameState and displays the results on the screen. This function renders only the top frame in frameState. See also **tsaHtmlRenderRenderAllFrames**.

Note that **tsaHtmlRenderRenderFrame** will automatically render the first hotspot if one exists in view, so the user does not have to make a call to **tsaHtmlRenderHotspot** after calling **tsaHtmlRenderRenderFrame**.

tsaHtmlRenderRenderAllFrames

```
tmLibappErr_t tsaHtmlRenderRenderAllFrames(
   Int   instance,
   Int   frameState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory was available.
`HTMLRENDER_ERR_NULL_GET_OBJECT_FUNC`	No 'get object' function is available.

Description

Renders the frame given by frameState and all of its subframes, and displays the results on the screen.

Note that **tsaHtmlRenderRenderAllFrames** automatically renders the first hotspot if one exists in view, so the user does not have to make a call to **tsaHtmlRenderHotspot** after calling **tsaHtmlRenderRenderAllFrames**.

tsaHtmlRenderRenderHotspot

```
tmLibappErr_t tsaHtmlRenderRenderHotspot(
   Int                        instance,
   Int                        frameState,
   tsaHtmlRenderHotspotDir_t  dir,
   Int                        hotspotId
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.
`dir`	The direction in which to find the next hotspot.
`hotspotId`	The ID of the hotspot to be rendered (in the case that **dir** is **tsaHtmlRenderHotspotSelect**).

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory available to allocate the history list for this page.
`HTMLRENDER_ERR_NO_HOTSPOT`	The current page contains no hotspots.
`HTMLRENDER_ERR_NO_HOTSPOT_ACTION`	No hotspot action was specified during instance setup.
`HTMLRENDER_ERR_INVALID_HOTSPOT_ID`	HotspotId is invalid.
`HTMLRENDER_ERR_INVALID_HOTSPOT_DIRECTION`	`dir` is invalid.

Description

The function renders the next hotspot according to the direction specified. If an alink is specified in the HTML page, the hotspot will show the alink color. If **flags** in the instance setup is **tsaHtmlRenderHotspotActionBorder**, then the hotspot will be highlighted with a box. See **tsaHtmlRenderSetupFlags_t** and **tsaHtmlRenderHotspotDir_t**.

tsaHtmlRenderGetFrameId

```
tmLibappErr_t tsaHtmlRenderGetFrameId(
   Int   instance,
   Int   frameState,
   Int  *frameID
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.
`frameID`	Pointer to a variable in which to return the frame ID.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Returns the frame ID associated with the given HtmlRender frame state.

tsaHtmlRenderGetCurrentHotspot

```
tmLibappErr_t tsaHtmlRenderGetCurrentHotspot(
   Int                          instance,
   Int                          frameState,
   tsaHtmlHotspotType_t         *hotspotType,
   char                         **url,
   ptsaHtmlRenderWidgetState_t  *widgetState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.
`hotspotType`	Pointer to a variable in which to return the hotspot type.
`url`	Pointer to a variable in which to return (a pointer to) the URL string.
`widgetState`	Pointer to a variable in which to return a pointer to the widget state associated with the current hotspot.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NO_HOTSPOT`	No hotspot was found.

Description

Returns the type, URL, and the widget state associated with the current hotspot.

Note that **url** is valid only for the hyperlinks and **widgetState** is valid only for widgets.

tsaHtmlRenderGetHotspot

```
tmLibappErr_t tsaHtmlRenderGetHotspot(
   Int                           instance,
   Int                           frameState,
   Int                           hotspotId,
   tsaHtmlHotspotType_t         *hotspotType,
   char                        **url,
   ptsaHtmlRenderWidgetState_t  *widgetState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frame state.
`hotspotId`	The ID of the hotspot to get.
`hotspotType`	Pointer to a variable in which to return the hotspot type.
`url`	Pointer to a variable in which to return (a pointer to) the URL string.
`widgetState`	Pointer to a variable in which to return a pointer to the widget state associated with the current hotspot.

Return Codes

`TMLIBAPP_OK`	Success.
`HTMLRENDER_ERR_INVALID_HOTSPOT_ID`	**hotspotId** is invalid.

Description

Returns the type, URL, and the widget state associated with the hotspot specified by hotspotId.

Note that **url** is only valid fot the hyperlinks and **widgetState** is only valid for widgets.

tsaHtmlRenderGetNumHotspots

```
tmLibappErr_t tsaHtmlRenderGetNumHotspots(
   Int     instance,
   Int     frameState,
   Int    *numHotspots
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frameState.
`numHotspots`	Pointer to a variable in which to return the number of hotspots in the current (parsed) HTML page.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Returns the number of hotspots in the current parsed HTML page.

tsaHtmlRenderGetSubFrame

```
tmLibappErr_t tsaHtmlRenderGetSubFrame(
   Int   instance,
   Int   frameState,
   Int   frameId,
   Int  *subFrameState
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frameState.
`frameId`	The ID of the frame to reference.
`subFrameState`	Pointer to a variable in which to return the subframe state.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Returns the subframe state associated with the given frame ID.

Note that the current HTML parser does not support frames. This function is provided for future extension.

tsaHtmlRenderGetNumSubFrames

```
tmLibappErr_t tsaHtmlRenderGetNumSubFrames(
   Int   instance,
   Int   frameState,
   Int  *numSubFrames
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frameState.
`numSubFrames`	Pointer to a variable in which to return the number of subframes in the current frame.

Return Codes

`TMLIBAPP_OK`	Successful.

Description

Gets the number of subframes in the current frame.

Note that the current HTML parser does not support frames. This function is provided for future extension.

tsaHtmlRenderFollowNamedLink

```
tmLibappErr_t tsaHtmlRenderFollowNamedLink(
   Int    instance,
   Int    frameState,
   char  *url
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frameState.
`url`	The named link.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`HTMLRENDER_ERR_NAMED_LINK_NOT_FOUND`	The URL was not found in list of named links.

Description

Follows a named link by scrolling the page (if the named location is not currently in view) and then selecting the first hotspot following the named location. This function should be called when the URL of a link hotspot contains #. Assuming the named location is on the same page, the page does not have to be parsed again.

tsaHtmlRenderScrollScreen

```
tmLibappErr_t tsaHtmlRenderScrollScreen(
   Int                        instance,
   Int                        frameState,
   tsaHtmlRenderScrollDir_t   direction
);
```

Parameters

`instance`	The instance, as returned by **tsaHtmlRenderOpen**.
`frameState`	The HtmlRender frameState.
`direction`	The direction in which to scroll the screen.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Instance not previously opened.
`TMLIBAPP_ERR_NOT_SETUP`	Instance not previously set up.
`HTMLRENDER_ERR_INVALID_SCROLL_DIRECTION`	**direction** is invalid.

Description

Scrolls the screen up or down according to **direction**. If there is no additional image in that direction, the function returns immediately. The first hotspot on the new part of the page then becomes the current hotspot.

The image scrolls up or down in increments of the screen size.

Chapter 7

Object Manager (OM) API

Topic	Page
Object Manager Overview	208
Object Manager API Data Structures	212
Object Manager API Enumerated Types	215
Object Manager API Functions	217

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

Object Manager Overview

The TriMedia Object Manager (OM) library provides a way to retrieve the objects from a database. The database must be built by an utility application called *Object Manager Database Builder* (OMDB). Currently, the database supports only three type of objects: HTML object, Image (both GIF and JPEG) object and Font (TM font only) object. The library complies with TriMedia Software Architecture (TSA).

Object Manager

The interface of the OM is very simple. There is only one other function besides the basic TSA functions. To request an object in the database, specify the location and the type of the object. The OM then returns a pointer to the object data structure. Memory needed for the object (including the necessary data structures) will have been allocated when the database is loaded. The OM does not allocate memory except for instance variables. It simply checks whether the requested object is in the database and returns a pointer to the object if found. Otherwise, the OM returns an error code.

To use the object manager, follow these basic steps:

1. Create an instance of the object manager library by calling **tsaOMtOpen**.
2. Call **tsaOMGetInstanceSetup** to get a copy of the instance setup structure. The field **ObjectData** in the setup structure should point to the data array generated by the OMDB (discussed below).
3. Complete the instance setup by calling **tsaOMSetupInstance**.

After obtaining a valid object manager instance, you can retrieve objects in the database by calling **tsaOMGetObject**.

Object Manager Database Builder

The object manager looks for information from an object database. You must build this database using a utility program called the Object Manager Database Builder (OMDB). The OMDB brings together HTML, image, and font objects. The database is in the form of a data file for host-assisted applications and an array of binary data for no-host application.

The role of the OMDB is to read a set of HTML pages, images or fonts, identify their data type from their file extension, decode the data if needed, allocate spaces for the object data structures and create a database from the resulting data. The object database is indexed by file name, and contains information describing the objects' type, size, font characteristics, and other attributes. The object manager returns a pointer to an object as a response to a select query.

The OM supports three types of objects:

- HTML pages (**tsaOMHTML_t**)
- Images (**tsa2DImage_t**)
- Fonts (**tsa2DFont_t**)

Encoded data such as GIF and JPEG images must be decoded to YUV (4:2:2) format before writing to the database. Font data must be loaded from raw data files. Each TM font has two associated files, a font bitmap file (*.bit) and a font metrics file (*.mtr).

Database Builder

For UNIX platforms, the database builder software is named omdb.out. Those who use Microsoft Windows 95/NT may use omdb.exe (note that a runtime DLL, cygwin1.dll, is required for the omdb.exe to run on Win95/NT).

The database builder assembles all files (currently it supports only files that have extensions htm, html, jpg, gif, bit and mtr) into one data array and adds directory information. As it traverses a sub-directory, it processes any supported file it finds and creates an associated database index, data structures and object data.

To run the utility, you must change to the directory where the data resources are located and then simply type the following at the command prompt:

```
omdb.exe pathname                    (under Windows)
```

where pathname is the absoulte path and name of the output database. You can also type

```
omdb.exe -h
```

at the command prompt to get the latest help information.

Assuming that the output database is named omdb, two output data files will be generated in the directory specified by pathname, omdb.dat and omdb_data.c. The omdb.dat is a binary data file and omdb_data.c is a C-language data array which can be compiled with other applications. The data array is for no-host applications in which all executable code and data reside in flash memory.

Database Loader

When the object manager sets up an OM instance, the database that has been loaded into memory is patched to improve access speed. First, the OM checks the version of the database, preventing the older version of the database from being used. You should always use the updated OMDB to build a new database. Second, it checks whether the database has been patched already. The database can be patched one time only because the loader patches the database by writing directly into the memory where the database is. If the database is patched, the loader has nothing to do, so it quits.

In the instance setup structure, there is a field **ObjectData** which is a pointer to **UInt8**. This field points to the data array (generated by omdb.exe). You need only assign the

address of the data array to the **ObjectData** pointer before the OM instance is set up. All other loading and patching processes are transparent.

Database Format

The object manager database consists of these areas:

- Database index area
- Object data structure area
- Raw data area
- URL string area
- Zero fill area

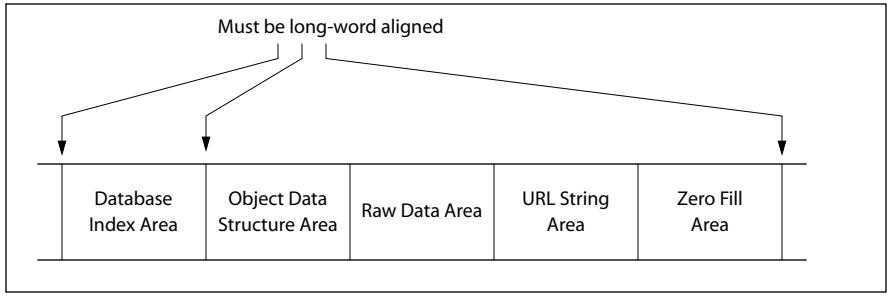


Figure 5 Database Object Format

Figure 5 illustrates a database object. The database index area is the object's header. It contains the type, size and URL of the object, addresses of the object data structure area and the next database object.

Depending on the complexity of the object, the data structure area can contain several layers of structural information. The object data itself is stored in the raw data area. An HTML object is stored as ASCII data. A decoded image is stored in YUV (4:2:2) format. A font bitmap is stored in binary form. The URL string area contains a null-terminated string. The format of a URL is

```
protocol:///<database path>/<object.type>
```

For instance, to specify an HTML page, an image object and a plain style TM font of size 14, the URLs could be

```
file:///html/trimedia.html
file:///images/trimedia.jpg
file:///fonts/plain14.font
```

Note that only file:// protocol is supported and the types of the object supported in URL are 'htm' or 'html' for HTML objects, 'gif' or 'jpg' for image objects and 'font' for font objects. The size of the database index and object data structure areas are multiple of 4

bytes. The other areas need not be aligned. To make sure that each object in the database is long-word aligned, the OM adds zeros at the end of the URL string area when needed.

Figure 6 and Figure 7 show the content of the object data structure area. The HTML object has one associated data structure. The data structure for an HTML object is discussed on page 214. An image object has 3 levels of hierarchy in its associated data structure. The data structure for an image object is discussed in Chapter 2, *2D Graphics API*.

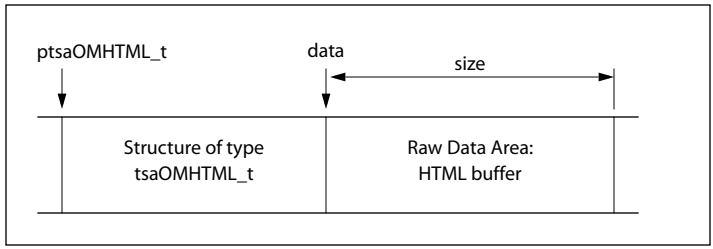


Figure 6 HTML Object Data Structure

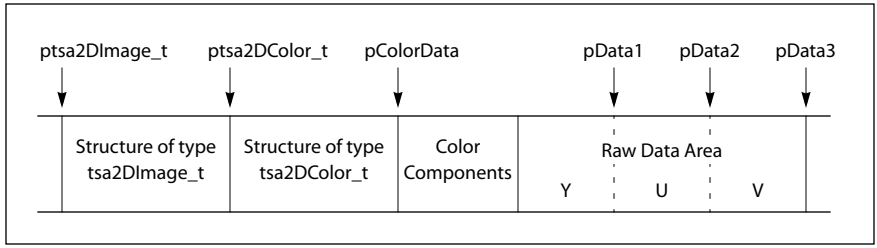


Figure 7 Image Object Data Structure

The data structure of font objects, shown in Figure 8, is more complicated than the other two object types. There are four levels of hierarchy in the object data structure area containing the font information and characteristics. You can find the details of each of these font-related data structures in Chapter 2, *2D Graphics API*.

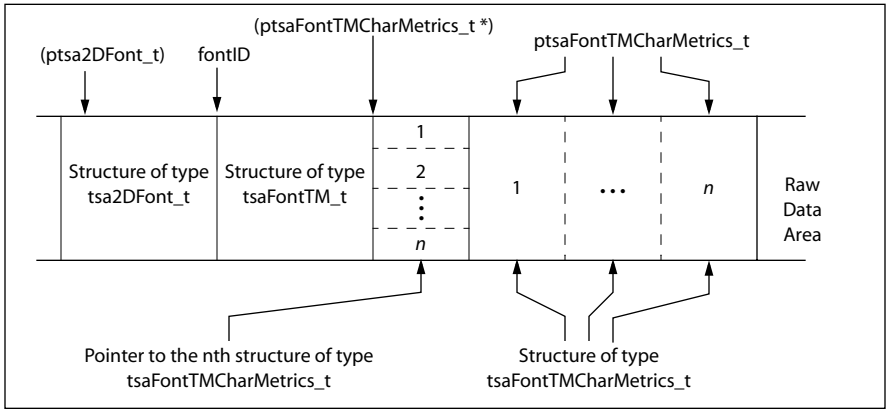


Figure 8 Font Object Data Structure

Object Manager API Data Structures

This section presents the Object Manager data structures.

Name	Page
tsaOMCapabilities_t	213
tsaOMInstanceSetup_t	213
tsaOMHTML_t	214

tsaOMCapabilities_t

```
typedef struct OMCapabilities {
   ptsaDefaultCapabilities_t   defaultCapabilities;
} tsaOMCapabilities_t, *ptsaOMCapabilities_t;
```

Fields

`defaultCapabilities`	Pointer to the default capabilities structure (see tsa.h)

Description

Describes the capabilities of the Object Manager. It does not have any capabilities other than those in **defaultCapabilities**.

tsaOMInstanceSetup_t

```
typedef struct OMInsatnceSetup {
   UInt8   *ObjectData;
} tsaOMInstanceSetup_t, *ptsaOMInstanceSetup_t;
```

Fields

`ObjectData`	Pointer to a data array (HTML pages, images and fonts) previously created by OMDB. The data array originated in the file omdb_data.c.

Description

Holds initial information from the application to set up the Object Manager instance. Only the one member of this structure must be set before setting up the Object Manager instance.

tsaOMHTML_t

```
typedef struct OMHTML {
   Char  *data;
   Int    size;
} tsaOMHTML_t, *ptsaOMHTML_t;
```

Fields

`data`	An HTML ASCII buffer.
`size`	Size of the HTML ASCII buffer.

Description

Holds the text of the HTML object from the database.

Object Manager API Enumerated Types

This section presents the (one) Object Manager enumerated type.

Name	Page
tsaOMType_t	216

tsaOMType_t

```
typedef enum {
   OM_TYPE_INVALID = 0,
   OM_TYPE_HTML,
   OM_TYPE_IMAGE,
   OM_TYPE_FONT
} tsaOMType_t;
```

Description

Enumerates the supported object types in the Object Manager. It is used in calls to **tsaOMGetObject** to specify the type of the requested object.

Object Manager API Functions

This section presents the Object Manager library functions.

Name	Page
tsaOMGetCapabilities	218
tsaOMOpen	218
tsaOMGetInstanceSetup	219
tsaOMInstanceSetup	220
tsaOMClose	221
tsaOMGetObject	222

tsaOMGetCapabilities

```
tmLibappErr_t tsaOMGetCapabilities(
   ptsaOMCapabilities_t   *cap
)
```

Parameters

`cap`	Pointer to a variable in which to return a pointer to the OM capabilities structure.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Gets the capabilities of the Object Manager.

tsaOMOpen

```
tmLibappErr_t tsaOMOpen(
   Int   *instance
)
```

Parameters

`instance`	Pointer (returned) to the instance.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NO_INSTANCE_AVAILABLE`	No instance available.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory to allocate for the instance data.

Description

Allocates an instance of the Object Manager. Initializes the instance setup structure to default values.

tsaOMGetInstanceSetup

```
tmLibappErr_t tsaOMGetInstanceSetup(
   Int                       instance,
   ptsaOMInstanceSetup_t   *setup
)
```

Parameters

`instance`	The instance, as returned by **tsaOMOpen**.
`setup`	Pointer to a variable in which to return a pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Returns an instance setup structure.

tsaOMInstanceSetup

```
tmLibappErr_t tsaOMInstanceSetup(
   Int                     instance,
   tsaOMInstanceSetup_t   *setup
)
```

Parameters

`instance`	The instance, as returned by **tsaOMOpen**.
`setup`	Pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.
`OM_ERR_NULL_DATABASE`	The database is empty.
`OM_ERR_DB_NOT_4_BYTES_ALIGNED`	The starting address of the database is not 4-byte aligned.
`OM_ERR_DB_VERSION_MISMATCHED`	The version of the database does not match that of the current OMDB. This error occurs if you pass an older database to the OM. Always use an updated OMDB to build the database.

Description

Sets up the instance of the Object Manager. If the database is empty, the function returns an error code. The function also checks whether the database is 4-byte aligned, has corrected database version, and is unpatched. If so, it patches the database. The OMDB produces a database with the same version number as the OMDB itself. The version of the OMDB can be found from the program banner when it is executed.

tsaOMClose

```
tmLibappErr_t tsaOMClose(
   Int   instance
)
```

Parameters

`instance`	The instance, as returned by **tsaOMOpen**.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Deallocates the instance previously open by **tsaOMOpen**. Frees all memory associated with the instance.

tsaOMGetObject

```
tmLibappErr_t tsaOMGetObject(
   Int           instance,
   Char         *url,
   tsaOMType_t   type,
   Pointer      *pObject
)
```

Parameters

instance	The instance, as returned by **tsaOMOpen**.
url	The name of the database object to be retrieved.
type	The type of the requested object.
pObject	Pointer to a variable in which to return a pointer to the object.

Return Codes

TMLIBAPP_OK	Success.
TMLIBAPP_ERR_NOT_SETUP	The Object Manager instance is not set up.
OM_ERR_OBJECT_NOT_IN_DATABASE	The requested object is not found in the database.
OM_ERR_UNSUPPORTED_URL_PROTOCOL	The protocol used in the URL is not supported. At present, only file:// protocol is supported.

Description

Get the object from the database by specifying its location and type. The function returns a pointer to the requested object.

Chapter 8

Widget API

Topic	Page
Introduction	224
Widget Library Overview	224
Widget Example Programs (exWidget) Overview	226
Widget Library Data Structures	227
Widget Library Functions	232
Standard Widget Set	238
Standard Widget Set Enumerated Types	239
Standard Widget Set Functions and Macros	250
How to Write Widgets	276

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

Introduction

Widgets are general-purpose graphic devices which can help you build a user interface. They also support TriMedia's HTML parser and renderer.

The widget library provides a framework in which you design the graphics (using TriMedia 2D graphics) of a widget and some or all of its behavior and the library takes care of operating the widgets.

The widget library provide these widget types, described later:

Button	Image	Toggle
Select menu	Slider (horizontal)	
Text area	Text line	Password

This chapter has three sections. The first section presents the concepts and the basic operations of the widget library. It tells you how to create and use a standard widget. It also presents the TSA data structures and API. An example widget program is also described here.

The second section describes the currently supported widget set. Each widget has its own specific attributes described by an enumerated type. This section also presents the widget creation functions. Although the implementations of the widgets are different, the widget creation functions are consistent.

The third section describes the TriMedia's Widget library framework, internal macros, and data structures. An example which can be used as a template to write new widgets can be found in the example/exWidgetTemplate directory. This section is for the users who want to implement their own widgets.

Widget Library Overview

Using the widgets library itself is very simple. After you create an instance of the widget library, you can create and use widget objects as needed. Widget attributes can be accessed any time by functions and macros.

Widgets created by this library have some common attributes:

- All widgets have pointers to plot, update, get value, and set value functions.
- All widgets have pointers to user-specific data.
- All widgets have a rectangular boundary (width and height, and coordinate location).
- All widgets point to their associated output packet.

Each widget also has its own specific attributes which differ from one widget to the next. Once a widget's attributes have been initialized (or changed), the widget can be rendered (or updated) to its associated packet.

Basic Operations

The basic widget operations include the following:

- Widget creation.
- Widget rendering.
- Get widget attributes.
- Set widget attributes.

After you have created a widget object, the widget's data structure contains both the common and widget-specific attributes. Function pointers in the common attributes define the widget-specific operations. These functions operate on the rest of the fields in the widget data structure. The widget data structure contains everything to implement a widget.

Thus, the interface to the widget library is comparatively simple. Each widget type has its own creation function. There is a single plot function, a single update function, and several get and set functions shared by all widgets.

How to Create a Standard Widget

1. Create an instance of the widget library by calling **tsaWidgetOpen.**
2. Call **tsaWidgetGetInstanceSetup** to get the instance setup structure.
3. Set up instances of the 2D Graphic library and the Object Manager library. They are required for the widget instance. In some cases, you might want to use your own widgets which do not use the Object Manage. In those cases there is no need to set up the instance of the Object Manager library.
4. Call **tsaWidgetSetupInstance** to complete the instance setup.

After obtaining a valid widget library instance, create and manipulate widgets by calling widget functions. For example, to create a button:

1. Call **tsaWidgetCreateButton** to create a button-specific data structure. A pointer to the data structure is returned.
2. Set widget-specific attributes, such as the text and border color for the button, using **tsaWidgetSet**.
3. Call **tsaWidgetPlot** to render the widget (to its associated packet).

If you want to change widget attributes (e.g., when you want to change the border color when the button is activated) call **tsaWidgetSet** with appropriate attribute index and new color, and then call **tsaWidgetPlot** or **tsaWidgetUpdate** to make the change.

Widget Example Programs (exWidget) Overview

Two example programs are provided. One of the example program can be found in example/exWidget/ directory. This example program demonstrates the use of the TriMedia Widget Library. Using the standard HTML Widget Set, this example program demonstrates what might be a possible Graphical User Interface (GUI) for Digital Television. This example program requires the 2D Graphics Library, the Widget Library and the Object Manager Library. The output is via TriMedia's Video Out.

Note
For simplicity, no anti-flicker filtering is done.

The demo is self-running and does not take any argument. Once the program is started, it will run thru the preprogrammed demos to completion. The demo starts with a background screen, followed by three different demo screens: the Login Screen demo, the Order Screen demo and the Audio Screen demo.

The other example is discussed in the *How to Write Widgets* on page 276.

Wrapper Function: myGetObject

In exWidget, the Object Manager library is used to get the objects from its database. In some cases, if the required object cannot be not found from the database, object manager returns an error code of **OM_ERR_OBJECT_NOT_FOUND** to the application. However, this scalar error message does not give enough information of which object (and its URL) was not found from the database. In order to resolve this problem, a wrapper function, **myGetObject**, is used instead of calling **tsaOMGetObject** directly. **myGetObject** actually calls **tsaOMGetObject** and prints the error code with the associated URL if error occurs.

The wrapper function can be found in the example /exWidget/Support.c.

Widget Library Data Structures

This section presents the widget library TSA data structures.

Name	Page
tsaWidgetCapabilities_t	228
tsaWidgetInstanceSetup_t	229
tsaWidgetInstVar_t	230
_tsaWidgetObject_t	231

tsaWidgetCapabilities_t

```
typedef struct {
   ptsaDefaultCapabilities_t   defaultCapabilities;
} tsaWidgetCapabilities_t, *ptsaWidgetCapabilities_t;
```

Fields

`defaultCapabilities`	Pointer to the default capabilities structure. (See tsa.h.)

Description

tsaWidgetCapabilities_t describes the capabilities of the Widget. It does not have any other capabilities data other than those in **defaultCapabilities**.

tsaWidgetInstanceSetup_t

```
typedef struct {
   Pointer         (*MallocFn)(size_t size);
   void            (*FreeFn)(Pointer pPtr);
   tmLibappErr_t   (*GetObjectFn)(
                     Int instance,
                     Char *url,
                     tsaOMType type,
                     Pointer *pObject );
   Int               tsa2DInst;
   Int               tsaOMInst;
} tsaWidgetInstanceSetup_t, *ptsaWidgetInstanceSetup_t;
```

Fields

`MallocFn`	Memory allocation function from the application.
`FreeFn`	A function that releases a previously allocated block of memory by **MallocFn**.
`GetObjectFn`	A callback function that retrieves an object (specified by both URL and type) from the Object Manager database. Typically, you would use awrapper function **myGetObject** which calls **tsaOMGetObject** if you are using the TriMedia Object Manager. You can set this field to null if you are not.
`tsa2DInst`	2D Graphics library instance.
`tsaOMInst`	Object Manager library instance.

Description

The structure holds initial information from the application to set up the Widget instance. All fields except **GetObjectFn** must be initialized by the application during setup. The callback function **GetObjectFn** is called indirectly to get objects from the database. Since you might want to have your own widgets which do not use the Object Manager, you can set **GetObjectFn** to null.

tsaWidgetInstVar_t

```
typedef struct WidgetInstVar {
   ptsaWidgetInstanceSetup_t   setup;
   tsaWidgetInstanceSetup_t    actual;
} tsaWidgetInstVar_t, *ptsaWidgetInstVar_t;
```

Fields

`setup`	Pointer to the instance setup structure.
`actual`	Actual instance setup structure.

Description

tsaWidgetInstVar_t is used for internal implementation only.

_tsaWidgetObject_t

```
struct _tsaWidgetObject_t {
   tmLibappErr_t   (* Plot)  (Int instance, tsaWidget_t widget);
   tmLibappErr_t   (* Update)(Int instance, tsaWidget_t widget);
   tmLibappErr_t   (* Get)   (Int instance, tsaWidget_t widget);
   tmLibappErr_t   (* Set)   (Int instance, tsaWidget_t widget);
   Int              instance;
   ptmAvPacket_t    pPcaket;
   Int              x;
   Int              y;
   Int              width;
   Int              height;
   Pointer          userData;
};
```

Fields

`Plot`	Function pointer to the widget plot function.
`Update`	Function pointer to the widget update function.
`Get`	Function pointer to the widget get function.
`Set`	Function pointer to the widget set function.
`instance`	The widget instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x,y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`userData`	Pointer to a user-specific data.

Description

tsaWidgetObject_t describes the common fields of a widget object. Note that users must not access this data structure directly. Instead, a set of macros is provided to access each field within the widget object.

Widget Library Functions

This section presents the widget library TSA (streaming architecture) functions.

Name	Page
tsaWidgetGetCapabilities	233
tsaWidgetOpen	234
tsaWidgetGetInstanceSetup	235
tsaWidgetInstanceSetup	236
tsaWidgetClose	237

tsaWidgetGetCapabilities

```
tmLibappErr_t tsaWidgetGetCapabilities(
   ptsaWidgetCapabilities_t   *cap
)
```

Parameters

`cap`	Pointer to a variable in which to return a pointer to the capabilities structure.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Gets the capabilities of the widget library.

tsaWidgetOpen

```
tmLibappErr_t tsaWidgetOpen(
   Int   *instance
)
```

Parameters

`instance`	Pointer to the (returned) instance.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NO_INSTANCE_AVAILABLE`	No instance available.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Not enough memory to allocate for instance variable.

Description

Allocates an instance of the widget library. Initializes the setup structure to default values.

tsaWidgetGetInstanceSetup

```
tmLibappErr_t tsaWidgetGetInstanceSetup(
   Int                          instance,
   ptsaWidgetInstanceSetup_t  *setup
)
```

Parameters

`instance`	The instance.
`setup`	Pointer to a variable in which to return a pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Gets the instance's setup structure.

tsaWidgetInstanceSetup

```
tmLibappErr_t tsaWidgetInstanceSetup(
   Int                          instance,
   tsaWidgetInstanceSetup_t   *setup
)
```

Parameters

`instance`	The instance.
`setup`	Pointer to the instance setup structure.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_NO_2D_INSTANCE`	2D Graphics instance is missing.
`WIDGET_ERR_NO_MALLOC_FUNC`	Memory allocation function is missing.
`WIDGET_ERR_NO_FREE_FUNC`	Memory free function is missing.

Description

Sets up the instance of the widget library. Checks for the presence of instance of the 2D Graphics library. Checks for the presence of callback functions for memory allocation.

tsaWidgetClose

```
tmLibappErr_t tsaWidgetClose(
   Int   instance
)
```

Parameters

`instance`	The instance.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Deallocates the instance previously open in **tsaWidgetOpen**. Frees all memory associated with the instance.

Standard Widget Set

This section presents the standard widget set. Currently, the widget library supports these widget types:

- Button

  A rectangular region that displays some text.

- Image

  A rectangular region that displays an image (identified by a URL).

- Password

  A rectangular region that supports text entry. The text entered is not echoed.

- Select menu

  A rectangular region that displays some textual choices and possibly allows scrolling to display choices hidden from view. End-users can select the items.

- Slider (horizontal)

  A rectangular region consisting of a left part, middle part, right part, and a slider. (A volume control bar is an example of a slider.)

- Text area

  A rectangular region that supports the entry of multi-line text.

- Text line

  A rectangular region that supports the entry of some text. The text entered is echoed.

- Toggle

  A rectangular region that functions like a check box or a radio button.

Each widget type has its own create function.

Use the plot and update functions plot and update a widget to its associated packet. Other functions and macros set or get some particular widget attribute.

To erase a widget, call **tsaWidgetFill** to fill it with the background color. To terminate a widget, call **tsaWidgetDestroy** which releases the memory allocated for the widget data structures.

Standard Widget Set Enumerated Types

This section presents the standard widget enumerated data type.

Name	Page
tsaWidgetButtonIndex_t	240
tsaWidgetImageIndex_t	241
tsaWidgetPasswordIndex_t	242
tsaWidgetSelectIndex_t	243
tsaWidgetSliderIndex_t	245
tsaWidgetTextareaIndex_t	246
tsaWidgetTextlineIndex_t	248
tsaWidgetToggleIndex_t	249

tsaWidgetButtonIndex_t

```
typedef enum {
   WIDGET_BUTTON_FONT  = 1,
   WIDGET_BUTTON_TEXT,
   WIDGET_BUTTON_FONTCOLOR,
   WIDGET_BUTTON_BACKGROUNDCOLOR,
   WIDGET_BUTTON_BORDERCOLORRIGHT,
   WIDGET_BUTTON_BORDERCOLORBOTTOM,
   WIDGET_BUTTON_BORDERCOLORLEFT,
   WIDGET_BUTTON_BORDERCOLORTOP
} tsaWidgetButtonIndex_t;
```

Fields

WIDGET_BUTTON_FONT	TM font for the button widget.
WIDGET_BUTTON_TEXT	Text for the button widget.
WIDGET_BUTTON_FONTCOLOR	Text color.
WIDGET_BUTTON_BACKGROUNDCOLOR	Color for the button.
WIDGET_BUTTON_BORDERCOLORRIGHT	Border color on the right side of the widget.
WIDGET_BUTTON_BORDERCOLORBOTTOM	Border color on the bottom of the widget.
WIDGET_BUTTON_BORDERCOLORLEFT	Border color on the left side of the widget.
WIDGET_BUTTON_BORDERCOLORTOP	Border color on the top of the widget.

Description

Enumerates the attributes specific to the button widget. All button attributes are initialized to null when a button widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with button colors is **ptsa2DColor_t**. The data type associated with fonts is **ptsa2DFont_t**. The data type associated with button text is **String**.

tsaWidgetImageIndex_t

```
typedef enum {
   WIDGET_IMAGE_URL_IMAGE = 1,
} tsaWidgetImageIndex_t;
```

Fields

`WIDGET_IMAGE_URL_IMAGE`	The URL of the image.

Description

Enumerates the (one) attribute is specific to the image widget. The URL is initialized to null when the widget is created. You must call **tsaWidgetSet** to set the URL before rendering the image widget.

The data type associated with the image URL is **String**.

tsaWidgetPasswordIndex_t

```
typedef enum {
   WIDGET_PASSWORD_FONT  = 1,
   WIDGET_PASSWORD_TEXT,
   WIDGET_PASSWORD_FONTCOLOR,
   WIDGET_PASSWORD_BACKGROUNDCOLOR,
   WIDGET_PASSWORD_BORDERCOLORBOTTOM,
   WIDGET_PASSWORD_BORDERCOLORLEFT,
   WIDGET_PASSWORD_BORDERCOLORTOP,
   WIDGET_PASSWORD_FIRSTCHAR,
   WIDGET_PASSWORD_SHOWCURSOR,
   WIDGET_PASSWORD_URL_CURSOR
} tsaWidgetPasswordIndex_t;
```

Fields

`WIDGET_PASSWORD_FONT`	TM font for the password widget.
`WIDGET_PASSWORD_TEXT`	Text for the password widget.
`WIDGET_PASSWORD_FONTCOLOR`	Text color.
`WIDGET_PASSWORD_BACKGROUNDCOLOR`	Color for the password23 box.
`WIDGET_PASSWORD_BORDERCOLORRIGHT`	Border color on the right side of the widget.
`WIDGET_PASSWORD_BORDERCOLORBOTTOM`	Border color on the bottom of the widget.
`WIDGET_PASSWORD_BORDERCOLORLEFT`	Border color on the left side of the widget.
`WIDGET_PASSWORD_BORDERCOLORTOP`	Border color on the top of the widget.
`WIDGET_PASSWORD_FIRSTCHAR`	The location of the first character in the text string to display. (Associated variable type: **Int**)
`WIDGET_PASSWORD_SHOWCURSOR`	Control the display of the text cursor. If True, show the cursor. (Associated variable type: **Bool**)
`WIDGET_PASSWORD_URL_CURSOR`	The URL of the text cursor image. (Associated variable type: **String**)

Description

Enumerates the attributes specific to the password widget. Currently, all the attributes are initialized to default values when the widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with colors is **ptsa2DColor_t**. The data type associated with fonts is **ptsa2DFont_t**. The data type associated with text (e.g., for a URL) is normally **String**, except for **WIDGET_SELECT_OPTION_VAL** which associates with **Char**.**

tsaWidgetSelectIndex_t

```
typedef enum {
   WIDGET_SELECT_NUM_VISIBLE  = 1,
   WIDGET_SELECT_FIRST_VISIBLE,
   WIDGET_SELECT_CURSOR_POS,
   WIDGET_SELECT_NUM_OPTIONS,
   WIDGET_SELECT_OPTION_VAL,
   WIDGET_SELECT_OPTION_SELECTED,
   WIDGET_SELECT_SHOWCURSOR,
   WIDGET_SELECT_FONT,
   WIDGET_SELECT_FONTCOLOR,
   WIDGET_SELECT_BACKGROUNDCOLOR,
   WIDGET_SELECT_SCROLLBARCOLOR,
   WIDGET_SELECT_HIGHLIGHTCOLOR,
   WIDGET_SELECT_BORDERCOLORRIGHT,
   WIDGET_SELECT_BORDERCOLORBOTTOM,
   WIDGET_SELECT_BORDERCOLORLEFT,
   WIDGET_SELECT_BORDERCOLORTOP,
   WIDGET_SELECT_URL_CURSOR_UP,
   WIDGET_SELECT_URL_CURSOR_DOWN,
   WIDGET_SELECT_URL_CURSOR_LEFT,
   WIDGET_SELECT_URL_CURSOR_RIGHT
} tsaWidgetSelectIndex_t;
```

Fields

`WIDGET_SELECT_NUM_VISIBLE`	Number of items displayed on the select menu. (Associated variable type: **Int**)
`WIDGET_SELECT_FIRST_VISIBLE`	Which line is currently at the top of the visible window. (Associated variable type: **Int**)
`WIDGET_SELECT_CURSOR_POS`	On which line the cursor is positioned. (Associated variable type: **Int**)
`WIDGET_SELECT_NUM_OPTION`	Number of options, total, for the select menu. (Associated variable type: **Int**)
`WIDGET_SELECT_OPTION_VAL`	Text string. One for each option. (Associated variable type: **Char****)
`WIDGET_SELECT_OPTION_SELECTED`	Whether this option been selected. (Associated variable type: **Bool***)
`WIDGET_SELECT_SHOWCURSOR`	If True, show the cursor at the selected item. (Associated variable type: **Bool**)
`WIDGET_SELECT_FONT`	TM font.
`WIDGET_SELECT_FONTCOLOR`	Text color.
`WIDGET_SELECT_BACKGROUNDCOLOR`	Color for the select menu.
`WIDGET_SELECT_SCROLLBARCOLOR`	Color for the scroll bar.
`WIDGET_SELECT_HIGHLOGHTCOLOR`	Color for the selected item.

WIDGET_SELECT_BORDERCOLORRIGHT	Border color on the right side of the widget.
WIDGET_SELECT_BORDERCOLORBOTTOM	Border color on the bottom of the widget.
WIDGET_SELECT_BORDERCOLORLEFT	Border color on the left side of the widget.
WIDGET_SELECT_BORDERCOLORTOP	Border color on the top of the widget.
WIDGET_SELECT_URL_CURSOR_UP	The URL of the cursor image.
WIDGET_SELECT_URL_CURSOR_DOWN	The URL of the cursor image.
WIDGET_SELECT_URL_CURSOR_LEFT	The URL of the cursor image.
WIDGET_SELECT_URL_CURSOR_RIGHT	The URL of the cursor image.

Description

Enumerates the attributes specific to the select menu widget. Currently, all the attributes are initialized to default values when the widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with colors is **ptsa2DColor_t**. The data type associated with fonts is **ptsa2DFont_t**. The data type associated with text (e.g., for a URL) is normally **String**, except for **WIDGET_SELECT_OPTION_VAL** which associates with **char****.

tsaWidgetSliderIndex_t

```
typedef enum {
   WIDGET_SLIDER_CUR_VAL  = 1,
   WIDGET_SLIDER_NUM_POS,
   WIDGET_SLIDER_URL_LEFT,
   WIDGET_SLIDER_URL_MIDDLE,
   WIDGET_SLIDER_URL_RIGHT,
   WIDGET_SLIDER_URL_TAB
} tsaWidgetSliderIndex_t;
```

Fields

`WIDGET_SLIDER_CUR_VAL`	The current value of the slider. (Associated variable type: **Int**)
`WIDGET_SLIDER_NUM_POS`	Number of levels available for the slider. (Associated variable type: **Int**)
`WIDGET_SLIDER_URL_LEFT`	The URL of the image of the left part of the slider.
`WIDGET_SLIDER_URL_MIDDLE`	The URL of the image of the middle part of the slider.
`WIDGET_SLIDER_URL_RIGHT`	The URL of the image of the right part of the slider.
`WIDGET_SLIDER_URL_TAB`	The URL of the image of the slider tab.

Description

Enumerates the attributes specific to the slider widget. Currently, all the attributes are initialized to default values when the widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with text (e.g., for a URL) is **String**.

tsaWidgetTextareaIndex_t

```
typedef enum {
   WIDGET_TEXTAREA_TEXT  = 1,
   WIDGET_TEXTAREA_ROWS,
   WIDGET_TEXTAREA_NUM_VISIBLE,
   WIDGET_TEXTAREA_FIRST_COL_CHAR,
   WIDGET_TEXTAREA_FIRST_ROW_CHAR,
   WIDGET_TEXTAREA_CURSOR_ROW,
   WIDGET_TEXTAREA_CURSOR_POS,
   WIDGET_TEXTAREA_SHOWCURSOR,
   WIDGET_TEXTAREA_URL_CURSOR,
   WIDGET_TEXTAREA_FONT,
   WIDGET_TEXTAREA_FONTCOLOR,
   WIDGET_TEXTAREA_BACKGROUNDCOLOR,
   WIDGET_TEXTAREA_BORDERCOLORRIGHT,
   WIDGET_TEXTAREA_BORDERCOLORBOTTOM,
   WIDGET_TEXTAREA_BORDERCOLORLEFT,
   WIDGET_TEXTAREA_BORDERCOLORTOP
} tsaWidgetTextareaIndex_t;
```

Fields

`WIDGET_TEXTAREA_TEXT`	Text string for the textarea widget.
`WIDGET_TEXTAREA_ROWS`	Total number of rows in the textarea. (Associated variable type: **Int**)
`WIDGET_TEXTAREA_NUM_VISIBLE`	Number of visible rows in the textarea widget. (Associated variable type: **Int**)
`WIDGET_TEXTAREA_FIRST_COL_CHAR`	The first column to be displayed. (Associated variable type: **Int**)
`WIDGET_TEXTAREA_FIRST_ROW_CHAR`	The first row to be displayed. (Associated variable type: **Int**)
`WIDGET_TEXTAREA_CURSOR_ROW`	The location of the cursor in the textarea. (Associated variable type: **Int**)
`WIDGET_TEXTAREA_CURSOR_POS`	The location (column position) of the cursor in the textarea. (Associated variable type: **Int**)
`WIDGET_TEXTAREA_SHOWCURSOR`	If true, display the cursor. (Associated variable type: **Bool**)
`WIDGET_TEXTAREA_URL_CURSOR`	The URL of the image of the cursor.
`WIDGET_TEXTAREA_FONT`	TM font for the textarea widget.
`WIDGET_TEXTAREA_FONTCOLOR`	Text color.
`WIDGET_TEXTAREA_BACKGROUNDCOLOR`	Color for the textarea box.
`WIDGET_TEXTAREA_BORDERCOLORRIGHT`	Border color on the right side of the widget.

`WIDGET_TEXTAREA_BORDERCOLORBOTTOM`	Border color on the bottom of the widget.
`WIDGET_TEXTAREA_BORDERCOLORLEFT`	Border color on the left side of the widget.
`WIDGET_TEXTAREA_BORDERCOLORTOP`	Border color on the top of the widget.

Description

Enumerates the attributes specific to the textarea widget. Currently, all the attributes are initialized to default values when the widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with colors is **ptsa2DColor_t**. The data type associated with fonts is **ptsa2DFont_t**. The data type associated with text (e.g., for a URL) is **String**.

tsaWidgetTextlineIndex_t

```
typedef enum {
   WIDGET_TEXTLINE_FONT  = 1,
   WIDGET_TEXTLINE_TEXT,
   WIDGET_TEXTLINE_FONTCOLOR,
   WIDGET_TEXTLINE_BACKGROUNDCOLOR,
   WIDGET_TEXTLINE_BORDERCOLORRIGHT,
   WIDGET_TEXTLINE_BORDERCOLORBOTTOM,
   WIDGET_TEXTLINE_BORDERCOLORLEFT,
   WIDGET_TEXTLINE_BORDERCOLORTOP,
   WIDGET_TEXTLINE_FIRSTCHAR,
   WIDGET_TEXTLINE_SHOWCURSOR,
   WIDGET_TEXTLINE_URL_CURSOR
} tsaWidgetTextlineIndex_t;
```

Fields

`WIDGET_TEXTLINE_FONT`	TM font for textline widget.
`WIDGET_TEXTLINE_TEXT`	Text on the textline widget.
`WIDGET_TEXTLINE_FONTCOLOR`	Text color.
`WIDGET_TEXTLINE_BACKGROUNDCOLOR`	Color for the textline box.
`WIDGET_TEXTLINE_BORDERCOLORRIGHT`	Border color on the right side of the widget.
`WIDGET_TEXTLINE_BORDERCOLORBOTTOM`	Border color on the bottom of the widget.
`WIDGET_TEXTLINE_BORDERCOLORLEFT`	Border color on the left side of the widget.
`WIDGET_TEXTLINE_BORDERCOLORTOP`	Border color on the top of the widget.
`WIDGET_TEXTLINE_FIRSTCHAR`	The location of the first character in the text string to be display. (Associated variable type: **Int**).
`WIDGET_TEXTLINE_SHOWCURSOR`	If true, display the text cursor. (Variable type: **Bool**).
`WIDGET_TEXTLINE_URL_CURSOR`	The URL of the text cursor image.

Description

Enumerates the attributes specific to the textline widget. Currently, all the attributes are initialized to default values when the widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with colors is **ptsa2DColor_t**. The data type associated with fonts is **ptsa2DFont_t**. The data type associated with text (e.g., for a URL) is **String**.

tsaWidgetToggleIndex_t

```
typedef enum {
   WIDGET_TOGGLE_CHECKED  = 1,
   WIDGET_TOGGLE_URL_ON,
   WIDGET_TOGGLE_URL_OFF
} tsaWidgetToggleIndex_t;
```

Fields

`WIDGET_TOGGLE_CHECKED`	The initial state of the toggle widget. (Associated variable type: **Bool**)
`WIDGET_TOGGLE_URL_ON`	The URL of the image for 'on'.
`WIDGET_TOGGLE_URL_OFF`	The URL of the image for 'off'.

Description

Enumerates the attributes specific to the toggle widget. Currently, all the attributes are initialized to default values when the widget is created. You must call **tsaWidgetSet** to set all these attributes before rendering the widget.

The data type associated with text (e.g., for a URL) is **String**.

Standard Widget Set Functions and Macros

This section presents the standard widget API functions and macros. A macro call can improve speed (at the expense of code size).

Name	Page
tsaWidgetCreateButton	252
tsaWidgetCreateImage	253
tsaWidgetCreatePassword	254
tsaWidgetCreateSelect	255
tsaWidgetCreateSlider	256
tsaWidgetCreateTextarea	257
tsaWidgetCreateTextline	258
tsaWidgetCreateToggle	259
tsaWidgetPlot	260
tsaWidgetPLOT (macro)	261
tsaWidgetUpdate	262
tsaWidgetUPDATE (macro)	263
tsaWidgetGet	264
tsaWidgetGET (macro)	265
tsaWidgetSet	266
tsaWidgetSET (macro)	267
tsaWidgetGetPacket	268
tsaWidgetSetPacket	268
tsaWidgetGetX	269
tsaWidgetSetX	269
tsaWidgetGetY	270
tsaWidgetSetY	270
tsaWidgetGetWidth	271
tsaWidgetSetWidth	271
tsaWidgetGetHeight	272
tsaWidgetSetHeight	272
tsaWidgetGetuserData	273

tsaWidgetSetuserData	273
tsaWidgetFill	274
tsaWidgetDestroy	275

tsaWidgetCreateButton

```
tmLibappErr_t tsaWidgetCreateButton(
   Int             instance,
   ptmAvPacket_t   pPacket,
   Int             x,
   Int             y,
   Int             width,
   Int             height,
   tsaWidget_t    *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the button.
`width`	Width of the button.
`height`	Height of the button.
`pWidget`	Pointer (returned) to the newly created button.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory allocation failed while creating the button data structure.

Description

Creates a button widget. First, it checks the validity of the widget library instance and the packet. If a null packet is found, the function returns an error code. Then it allocates memory for the button-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the button data structure to the application.

tsaWidgetCreateImage

```
tmLibappErr_t tsaWidgetCreateImage(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer (returned) to the newly created image object.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory allocation failed while creating the image data structure.

Description

Creates an image widget. First, it checks the validity of the widget packet. If a null packet is found, the function returns an error code. Then it allocates memory for the image-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetCreatePassword

```
tmLibappErr_t tsaWidgetCreatePassword(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer (returned) to the newly created password.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_INVALID_INSTANCE`	Invalid input instance found.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory allocation failed while creating the password data structure.

Description

Creates a password widget. First, it checks the validity of the widget library instance and the packet. If a null packet is found, the function returns an error code. Then it allocates memory for the password-specific data structure. The common widget attributes are initialized at this time, whereas the specific widget attributes will be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetCreateSelect

```
tmLibappErr_t tsaWidgetCreateSelect(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer (returned) to the newly created select menu.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory allocation failed while creating the select menu data structure.

Description

Creates an select menu. First, it checks the validity of the widget packet. If a null packet is found, the function returns an error code. Then it allocates memory for the select-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetCreateSlider

```
tmLibappErr_t tsaWidgetCreateSlider(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is plotted.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer to the address of the newly created widget object.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory problem while creating slider widget data structure.

Description

Creates a slider widget. First, it checks the validity of the widget packet. If an invalid instance or a null packet is found, the function returns an error code. Then it allocates memory for the slider-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetCreateTextarea

```
tmLibappErr_t tsaWidgetCreateTextarea(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer (returned) to the newly created textarea.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory allocation failed while creating the textarea data structure.

Description

Creates a textarea widget. First, it checks the validity of the widget packet. If a null packet is found, the function returns an error code. Then it allocates memory for the textarea-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetCreateTextline

```
tmLibappErr_t tsaWidgetCreateTextline(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer (returned) to the newly created textline.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory allocation failed while creating the textline data structure.

Description

Creates a textline widget. First, it checks the validity of the widget packet. If a null packet is found, the function returns an error code. Then it allocates memory for the textline-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetCreateToggle

```
tmLibappErr_t tsaWidgetCreateToggle(
   Int            instance,
   ptmAvPacket_t  pPacket,
   Int            x,
   Int            y,
   Int            width,
   Int            height,
   tsaWidget_t   *pWidget
)
```

Parameters

`instance`	The instance.
`pPacket`	Pointer to a packet where the widget is rendered.
`x, y`	Coordinate of the widget.
`width`	Width of the widget.
`height`	Height of the widget.
`pWidget`	Pointer (returned) to the newly created widget object.

Return Codes

`TMLIBAPP_OK`	Success.
`TMLIBAPP_ERR_NULL_PACKET`	Null input packet pointer found.
`TMLIBAPP_ERR_MEMALLOC_FAILED`	Memory problem while creating toggle widget data structure.

Description

Creates a toggle widget. First, it checks the validity of the widget packet. If a null packet is found, the function returns an error code. Then it allocates memory for the toggle-specific data structure. The common widget attributes are initialized at this time whereas the specific widget attributes are going to be set later using **tsaWidgetSet**. The function returns a pointer to the widget data structure to the application.

tsaWidgetPlot

```
tmLibappErr_t tsaWidgetPlot(
   Int           instance,
   tsaWidget_t   widget
)
```

Parameters

`instance`	The instance.
`widget`	The widget data structure.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_NULL_WIDGET`	The widget was not created.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is invalid.

Description

The function plots (renders) a widget using the widget-specific plot function placed in the data structure when the widget was created.

Before you call **tsaWidgetPlot**, you must initialize all the widget-specific attributes in the data structure using **tsaWidgetSet**.

tsaWidgetPLOT

```
#define tsaWidgetPLOT( inst, w )
   ((struc _tsaWidgetObject *)w)->Plot((inst),(w)))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is invalid.
`WIDGET_ERR_NULL_GET_OBJECT_FUNC`	The function to the pointer to the get object is invalid.

Description

A macro version of **tsaWidgetPlot**.

tsaWidgetUpdate

```
tmLibappErr_t tsaWidgetUpdate(
   Int           instance,
   tsaWidget_t   widget
)
```

Parameters

`instance`	The instance.
`widget`	The widget data structure.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_NULL_WIDGET`	The widget is not created.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.
`WIDGET_ERR_NULL_GET_OBJECT_FUNC`	The function pointer to the get object function is invalid.

Description

The function updates (renders) the widget using the widget-specific update function placed in the data structure when the widget was created. After changing widget-specific attributes, you can call this function to redisplay the changes.

tsaWidgetUPDATE

```
#define tsaWidgetUPDATE( inst, w )
   (((struc _tsaWidgetObject *)w)->UPDATE((inst),(w)))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.
`WIDGET_ERR_NULL_GET_OBJECT_FUNC`	The function pointer to the get object function is invalid.

Description

A macro version of **tsaWidgetUpdate**.

tsaWidgetGet

```
tmLibappErr_t tsaWidgetGet(
   Int           instance,
   tsaWidget_t   widget,
   Int           index,
   Pointer      *pvalue
)
```

Parameters

instance	The instance.
widget	The widget data structure.
index	Index of the widget attribute.
pvalue	Pointer to a variable in which to return a pointer to the value of requested attribute.

Return Codes

TMLIBAPP_OK	Success.
WIDGET_ERR_NULL_WIDGET	The widget is not created.
WIDGET_ERR_INCORRECT_INSTANCE	The widget instance is incorrect.
WIDGET_ERR_INVALID_INDEX	The index of the widget attribute is invalid

Description

Gets a widget attribute. The function uses the widget-specific get function placed in the data structure when the widget was created. Refer to *Standard Widget Set Enumerated Types* starting on page 239 for specific attributes.

tsaWidgetGET

```
#define tsaWidgetGET( inst, w, i, p )
   (((struc _tsaWidgetObject *)w)->Get((inst),(w),(i),(p)))
```

Parameters

`instance`	The instance.
`w`	Pointer to the widget data structure.
`i`	Index of the widget attribute.
`p`	Pointer to a variable in which to return a pointer to the value of requested attribute.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.
`WIDGET_ERR_INVALID_INDEX`	The index of the widget attribute is invalid.

Description

A macro version of **tsaWidgetGet**.

tsaWidgetSet

```
tmLibappErr_t tsaWidgetSet(
   Int          instance,
   tsaWidget_t  widget,
   Int          index,
   Pointer      value
)
```

Parameters

`instance`	The instance.
`widget`	The widget data structure.
`index`	Index of the widget attribute.
`value`	Pointer to the value for the attribute to be changed.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_NULL_WIDGET`	The widget is not created.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.
`WIDGET_ERR_INVALID_INDEX`	The index of the widget attribute is invalid.

Description

Sets a widget attribute. The function uses the widget-specific set function placed in the data structure when the widget was created. Refer to *Standard Widget Set Enumerated Types* starting on page 239 for specific attributes.

tsaWidgetSET

```
#define tsaWidgetSET( inst, w, i, p )
   (((struc _tsaWidgetObject *)w)->Set((inst),(w),(i),(p)))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`i`	Index of the widget attribute.
`p`	Pointer to the value of the attribute to be changed.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.
`WIDGET_ERR_INVALID_INDEX`	The index of the widget attribute is invalid

Description

A macro version of **tsaWidgetSet**.

tsaWidgetGetPacket

```
#define tsaWidgetGetPacket( inst, w )
   (((struc _tsaWidgetObject *)w)->pPacket)
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

None.

Description

This macro gets the packet pointer from the widget data structure.

tsaWidgetSetPacket

```
#define tsaWidgetSetPacket( inst, w, v )
   (((struc _tsaWidgetObject *)w)->pPacket = (v))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`v`	Pointer to a packet of type **tmAvPacket_t**.

Return Codes

None.

Description

This macro sets the packet pointer in the widget data structure.

tsaWidgetGetX

```
#define tsaWidgetGetX( inst, w )
   (((struc _tsaWidgetObject *)w)->x)
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

None.

Description

This macro gets the current *x* coordinate of the widget.

tsaWidgetSetX

```
#define tsaWidgetSetX( inst, w, v )
   (((struc _tsaWidgetObject *)w)->x = (v))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`v`	Value of the *x* coordinate.

Return Codes

None.

Description

This macro sets the current *x* coordinate of the widget.

tsaWidgetGetY

```
#define tsaWidgetGetY( inst, w )
   (((struc _tsaWidgetObject *)w)->y)
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

None.

Description

This macro gets the current *y* coordinate of the widget.

tsaWidgetSetY

```
#define tsaWidgetSetY( inst, w, v )
   (((struc _tsaWidgetObject *)w)->y = (v))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`v`	Value of the *y* coordinate.

Return Codes

None.

Description

This macro sets the current *y* coordinate of the widget.

tsaWidgetGetWidth

```
#define tsaWidgetGetWidth( inst, w )
   (((struc _tsaWidgetObject *)w)->width)
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

None.

Description

This macro gets the width of the widget.

tsaWidgetSetWidth

```
#define tsaWidgetSetWidth( inst, w, v )
   (((struc _tsaWidgetObject *)w)->width = (v))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`v`	Width of the widget.

Return Codes

None.

Description

This macro sets the width of the widget.

tsaWidgetGetHeight

```
#define tsaWidgetGetHeight( inst, w )
   (((struc _tsaWidgetObject *)w)->height)
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

None.

Description

This macro gets the height of the widget.

tsaWidgetSetHeight

```
#define tsaWidgetSetHeight( inst, w, v )
   (((struc _tsaWidgetObject *)w)->height = (v))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`v`	Height of the widget.

Return Codes

None.

Description

This macro sets the height of the widget.

tsaWidgetGetuserData

```
#define tsaWidgetGetuserData( inst, w )
   (((struc _tsaWidgetObject *)w)->userData)
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.

Return Codes

None.

Description

This macro gets the pointer to the user-specific data from the widget data structure.

tsaWidgetSetuserData

```
#define tsaWidgetSetuserData(inst, w, v)
   (((struc _tsaWidgetObject *)w)->userData = (v))
```

Parameters

`inst`	The instance.
`w`	Pointer to the widget data structure.
`v`	Pointer to the user data buffer.

Return Codes

None.

Description

This macro sets the pointer to the user-specific data.

tsaWidgetFill

```
tmLibappErr_t tsaWidgetFill(
   Int             instance,
   tsaWidget_t     widget,
   ptsa2DColor_t   pColor
)
```

Parameters

`instance`	The instance.
`widget`	The widget data structure.
`pColor`	Pointer to a color.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_NULL_WIDGET`	The widget is not created
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.

Description

The function erases the widget with the background color.

tsaWidgetDestroy

```
tmLibappErr_t tsaWidgetDestroy(
   Int           instance,
   tsaWidget_t   widget
)
```

Parameters

`instance`	The instance.
`widget`	The widget data structure.

Return Codes

`TMLIBAPP_OK`	Success.
`WIDGET_ERR_NULL_WIDGET`	The widget is not created
`WIDGET_ERR_INCORRECT_INSTANCE`	The widget instance is incorrect.

Description

The function frees the memory for the widget object.

How to Write Widgets

The TriMedia Widget Library can be easily extended by adding user-designed widgets. In this section, the widget library framework, C header files and internal macros are described. These materials are needed when users want to implement their own widgets.

Widget Library Framework

The widget library is designed in the object-oriented approach. Each widget has its own private data structure which includes common and specific fields. The common fields have a set of function pointers (which define the widget implementation and operate only on the rest of the fields in the data structure), the instance, the geometry of the widget and the pointer to the user data. The specific fields are widget specific and depend on the widget implementation. When **tsaWidgetCreateXXX** is called, a pointer to the data structure of the **widgetXXX** is returned.

The widget library framework is defined in two header files: tsaWidget.h and tsaWidget-Internal.h. tsaWidget.h defines a common widget object fields (see page 231). tsaWidget-Internal.h defines the macros for the widget internal implementation that needs to access the widget data structure. The following table lists the macros defined in tsa-WidgetInternal.h.

WIDGET_DEFAULT_FIELDS	Declaration of the default widget fields in the private internal widget data structure.
WIDGET_FILL_DEFAULT_FIELDS	Fill in the default values to the widget common fields when **tsaWidgetCreateXXX** is called.
WIDGET_CHECK_INSTANCE	Check the existence and validity of the input widget instance.This macro makes sure that the input widget instance is associated with the widget being used.
WIDGET_GET_PACKET	Macro to get the widget associated packet.
WIDGET_GET_X	Macro to get the x coordinate of the widget.
WIDGET_GET_Y	Macro to get the y coordinate of the widget.
WIDGET_GET_WIDTH	Macro to get the width of the widget.
WIDGET_GET_HEIGHT	Macro to get the height of the widget.
WIDGET_2DINST	Macro to get the instance of the 2D Graphics.
WIDGET_OMINST	Macro to get the instance of the Object Manager
WIDGET_GETOBJECT	Macro to get the function pointer of the get object function.

WIDGET_MALLOC	Macro to call the application-specific memory allocation function.
WIDGET_FREE	Macro to call the application-specific memory free function.

Widget Example (WidgetTemplate) Overview

The TextBox example program can be found in example/exWidgetTemplate/ directory. This example shows you how to write a widget that complies with the widget library framework. The details will be discussed in section three in this chapter. This example program requires the 2D Graphics Library and the Widget Library. This is no need of the Object Manager Library. The output is via TriMedia's Video Out.

Note
For simplicity no anti-flicker filtering is done. The demo is self-running and does not take any argument. Once the program is started, it shows a TextBox widget (3D look) on the screen, followed by another TextBox widget (Windows look) and the demo is done.

A TextBox widget is provided as an example to show how to write a widget module that works with the framework. This example contains twelve files. One is the Makefile and six of them are supportive modules: Color.c, Color.h, Font.c, Font.h, Support.c and Support.h. exWidgetTemplate.c is the main program.

There are four files for the TextBox widget:

- WidgetTextBox.h.
- WidgetTextBoxInternal.h.
- WidgetTextBox.c.
- WidgetTextBox2.c.

WidgetTextBox.h

WidgetTextBox.h defines the widget attribute indices, the public widget creation API, and the error codes. The figure below shows the TextBox attribute indices enumeration, **tsaWidgetTextBoxIndex_t**. This enumerates the possible widget-specific fields of the TextBox widget. Note that each index should have an associated widget specific field in the private widget data structure except **WIDGET_TEXTBOX_INVALID**.

```
typedef enum{
/* index */                    /* argument type */
   WIDGET_TEXTBOX_INVALID = 0
   WIDGET_TEXTBOX_TEXT,        /*string */
   WIDGET_TEXTBOX_BGCOLOR,     /* ptsa2DColor_t */
   WIDGET_TEXTBOX_FGCOLOR,     /* ptsa2DColor_t */
   WIDGET_TEXTBOX_FONT         /* ptsa2DFont_t */
} tsaWidgetTextBoxIndex_t;
```

WidgetTextBoxInternal.h

WidgetTextBoxInternal.h defines the private data structure of the TextBox widget. This header file is used only for the widget implementation, so it is not public. The data structure contains two parts: standard widget object fields and the widget-specific fields. The standard widget fields are declared via the macro **WIDGET_DEFAULT_FIELDS** and the widget specific fields follows. The figure below shows the TextBox widget data structure.

```
typedef struct {
/* Standard widget object fields */
   WIDGET_DEFAULT_FIELDS();
/* Widget specific fields */
   String             text;
   ptsa2DColor_t      bgcolor;
   ptsa2DColor_t      fgcolor;
   ptsa2DFont_t       font;
} TextBox, *pTextBox;
```

WidgetTextBox.c and WidgetTextBox2.c

WidgetTextBox.c contains the widget initialization, attributes access and implementation modules. There are four functions:

- **tsaWidgetCreateTextBox**
- **TextWidgetGet**
- **TextWidgetSet**
- **TextWidgetPlot**

Also, WidgetTextBox2.c is provided as a second TextBox based on the same data structure (same header files). The only difference between these two widgets is in the TextPlot function whereas the other three functions remain the same.

tsaWidgetCreateTextBox

tsaWidgetCreateTextBox is a public function used to allocate the memory for the TextBox data structure. Checking on the widget instance and the packet are needed. The macro **WIDGET_FILL_DEFAULT_FIELDS** is used to fill the common widget fields using the get/set value, plot/update functions, and the input parameters. Users can set some default values to the widget-specific attributes. A pointer to the TextBox widget data structure is returned to the application.

TextBoxGet

TextBoxGet is a static function which is used to report the current value of the widget specific attribute with appropriate attribute index.

TextBoxSet

TextBoxSet is a static function which is used to set the value of the widget specific attribute with appropriate attribute index.

TextBoxPlot

TextBoxPlot is also a static function which is a widget implementation module. It defines the widget appearance, accesses the current values of the widget attributes from the data structure, and renders the widget to its associated packet. Since the widget library is just a general-purpose graphic device, it only reflects the status of the system by means of visual effects. No aspect of system control issue is implemented in the plot function. In this example, both the **Plot** and **Update** function pointers in the data structure are assigned to the address of the **TextBoxPlot** function. To increase the rendering speed for updating the widget, the **TextBoxUpdate** function may be needed to modify only the part that is changed instead of drawing the whole widget again.

Chapter 9

Window Manager (WM) API

Topic	Page
Introduction	282
Windows	282
Returned Error Messages	286
Window Manager API Data Structures	287
Window Manager API Functions	291

Note
This component library is available as a part of the TriMedia DTV software system. It is not included with the basic TriMedia SDE, but it is available under a separate licensing agreement. Please contact your TriMedia sales representative for more information.

Introduction

The TriMedia Window Manager is a TSA compliant library that manages windows from multiple users. TSA compliant means that the structure of the API (like to opening and setting up an instance before use), and the main types of data structures (**tmAvPacket_t**, etc.) are shared with other TSA compliant software.

The window manager is limited in its functionality to make sure the performance and memory requirements are acceptable for a broad range of embedded applications. The main users that were kept in mind when defining the functionality are close captioning, OSD, and web browser. The first version window manager manages graphics only, and is not capable of handling a real-time video stream. To keep the window manager simple and general, the 2D library is used whenever possible to draw or 'blt' something to a buffer, and the displaying of the resulting composition of windows is left to the application (which can use the video renderer or a mechanism of its own).

Windows

Windows are rectangular and are ordered so that for every two windows, one of the two is higher than the other. A higher window obscures a lower window if they overlap. A window is created using the function **tsaWMCreateRealWindow** or **tsaWMCreateVirtualWindow**.

Window Types

The window manager supports several types of windows.

The most straightforward type is the real window. In this case, the user keeps a complete image of the window in memory so that the window manager can at any time copy those parts that are visible (the *cliplist*) to the backplane. This is fast and simple, but can be memory consuming.

A second type of window is the virtual window. In this case, the window manager calls a callback function, provided by the user, for every rectangle in the cliplist. The callback function has as arguments the ID of the window, the rectangle in the form of a complete packet, the rectangle in coordinates relative to the window, and the window in coordinates relative to the parent. The packet can be given to any 2D function to draw something, while the coordinates of the clip can be used to adjust the location of where to draw to fit the packet of the clip. For details on how to adjust, see the example program exWM.c.

Instances

Before the user can create a window, it needs to obtain and setup an instance using the function **tsaWMOpen** and **tsaWMInstanceSetup**. The instance is a parameter in every

WM function that manipulates a window, and is checked to see the window is manipulated by the user who created it. Windows of instances stick together, i.e. for every two instances A and B, all windows of A will be on top of all windows of B, or the other way around.

Video Out

The window manager does not provide the functionality to actually output the assembled windows to video-out. Instead, the first instance has to provide a tmAvPacket_t that contains a buffer that is used as the backplane. The color of the upper left pixel is taken as the background color. The window manager copies those parts of the windows that are visible to the backplane, and restores the backplane whenever the background becomes visible. The user is responsible for displaying the buffer. This provides more flexibility in video formats, as the window manager does not need to understand how to display the buffer. Note that the user cannot delete the tmAvPacket_t that holds the backplane buffer until all instances are closed.

Note
The WM does an optimized cache copyback after it updates the backplane.

Redrawing

To change the contents of a real window, the in-memory buffer has to be updated after which the function **tsaWMRedraw** can be called. This will force the window manager to update the backplane in those areas that are occupied by that window.

For a **virtual** window, the function **tsaWMRedraw** will in turn call the window specific callback function, to update those parts of the backplane that the window occupies directly. The cliplist of a window is not given out directly and should not be 'remembered', because it can change by WM calls by other users.

Moving

The function **tsaWMMoveWindow** can be used to move a window. The size of window will remain the same and the parts of other windows or the background that become visible are updated automatically. Every window is clipped to the size of the backplane, and to its parent, if it has one. Child windows will move with the parent; the location of a child window is relative to the parent window.

Stacking Order

As mentioned, windows of the same instance stick together. To raise or lower a window relative to the other windows of the instance **tsaWMRaiseWindow** and **tsaLWMowerWindow** can be used. To raise or lower (all windows of) an instance, **tsaWMRaiseAllWindows** and **tsaWMLowerAllWindows** can be used. When setting up an instance or creating a win-

dow, a stacking order can be defined to tell the window manager that an instance or window should always stay on top (**wmSO_ALWAYS_ON_TOP**) or bottom (**wmSO_ALWAYS_ON_BOTTOM**) unaffected by calls to the raise and lower functions by any user.

Display and Hiding

When a window is created, it is not visible yet. A non-visible window can be displayed using a call to **tsaWMDisplayWindow**. A visible window can be hidden by a call to **tsaWMHideWindow**. Whenever a window is hidden or displayed, other windows and the backplane are updated when needed.

Scrolling

Scrolling can be implemented by changing the *viewing window* of a real window. The viewing window is the part of the in memory buffer that is visible. It can be set using the function **tsaWMChangeViewingWindow**. By making the viewing window smaller than the size of the picture in the buffer, the offset can be varied to display different parts of the window without changing the part of the backplane that is occupied. The default viewing window is the complete window.

The concept of viewing window does not exist for virtual windows, so scrolling of virtual windows has to be implemented in the user provided callback function. The window manager does not offer a mechanism to repeatedly update the offset of the viewing window. So to scroll in a number of small steps after regular time intervals, it needs to be implemented by the user.

Locking by User

Windows can be locked and unlocked using **tsaWMLockWindow** and **tsaWMUnlockWindow**. When a window is locked, its image on the backplane is no longer updated. Locking a window gives the user the opportunity to update an in memory buffer without worrying that the window manager displays an incomplete image. When a window is unlocked, the backplane will be updated automatically if the window has been changed.

Reentrancy

In this release, all functions (except **tsaWMGetCapabilities**) suspend task scheduling. This effectively prevents other tasks from changing the internal data structures of the window manager.

Note
Implementation of reentrancy is now independent of tmos.

Parent Windows

A window can be created in a parent window by passing in the ID of the parent window at creation. A child of a parent will always be clipped to the viewing window of the parent. A child will always lay on top of the parent. Whenever the parent becomes invisible, the children become invisible too, but not the other way around. The child has a certain position relative to the upper left corner of the parent. For a real parent window with a viewing window different from the in memory buffer, the location of the child is relative to the upper left corner of the in memory buffer, i.e., when a parent scrolls up, the child moves up too.

Returned Error Messages

The following error messages are returned for the corresponding function.

Error code	API
WM_ERR_INVALID_INSTANCE	none
WM_ERR_INVALID_WINDOW_ID	tsaWMDestroyWindow, tsaWMRaiseWindow, tsaWMLowerWindow, tsaWMMoveWindow, tsaWMDisplayWindow, tsaWMHideWindow, tsaWMRedrawWindow, tsaWMChangeViewing-Window, tsaWMLockWindow, tsaWMUnlock-Window, tsaWMSetAbsoluteStackingOrder
WM_ERR_INVALID_PARENT_WINDOW	tsaWMCreateVirtualWindow, tsaWMCreateReal-Window
WM_ERR_NOT_CAPABLE	tsaWMInstanceSetup, tsaWMCreateRealWindow
WM_ERR_OVERLAPPING_TOPWINDOWS	none
WM_ERR_NOT_OWNER	tsaWMDestroyWindow, tsaWMRaiseWindow, tsaWMLowerWindow, tsaWMMoveWindow, tsaWMDisplayWindow, tsaWMHideWindow, tsaWMRedrawWindow, tsaWMChangeViewing-Window, tsaWMLockWindow, tsaWMUnlock-Window, tsaWMSetAbsoluteStackingOrder
WM_ERR_NOT_OWNER_PARENT	tsaWMCreateVirtualWindow, tsaWMCreateReal-Window
WM_ERR_INCORRECT_COORDINATES	none
WM_ERR_MEMORY	All WM APIs
WM_ERR_INVALID_STACKING_ORDER	tsaWMInstanceSetup, tsaWMCreateVirtual-Window, tsaWMCreateRealWindow, tsaWMSet-AbsoluteStackingOrder
WM_ERR_NOT_AVAILABLE_RIGHT_NOW	All WM API functions.
WM_ERR_INCORRECT_ARGUMENT	tsaWMInstanceSetup, tsaWMGetInstanceSetup, tsaWMCreateRealWindow
WM_ERR_NOT_IMPLEMENTED	tsaWMCreateRealWindow
WM_ERR_FORMAT_MISMATCH	tsaWMInstanceSetup, tsaWMCreateRealWindow

Window Manager API Data Structures

This section presents the Window Manager data structures.

Name	Page
tsaWMStackingOrder_t	288
ptsaRedrawCallbackFun_t	289
tsaWMCapabilities_t	290
tsaWMInstanceSetup_t	290

tsaWMStackingOrder_t

```
typedef enum {
   wmSO_NONE               = 0,
   wmSO_DONT_CARE          = 1,
   wmSO_ALWAYS_ON_TOP      = 2,
   wmSO_ALWAYS_ON_BOTTOM   = 3
} tsaWMStackingOrder_t, *ptsaWMStackingOrder_t ;
```

Fields

`wmSO_NONE`	Not used.
	Stacking order controlled by the functions **tsaWMRaiseWindow**, **tsaWMLowerWindow**, **tsaWMRaiseAllWindows**, and **tsaWMLowerAllWindows**.
	An instance or window must be on top of others at all times.
	An instance or window has to be on the bottom of others at all times.

Description

This type is used to tell the window manager the stacking order of windows.

ptsaRedrawCallbackFun_t

```
typedef  tmLibappErr_t (*ptsaRedrawCallbackFun_t) (
   UInt32           windowID,
   ptsa2DRect_t     pWindowRect,
   ptsa2DRect_t     pClipInWindow,
   tsa2DContext_t   pGrContext,
   tmAvPacket_t     pPacket
) ;
```

Fields

`windowID`	The ID of the window to be updated.
`pWindowRect`	Complete window relative to parent.
`pClipInWindow`	Current clip in window to be updated in this call.
`pGrContext`	The 2D lib graphical context of the window.
`pPacket`	A packet that contains exactly the part of the window that needs to be updated.

Description

Type of callback function that is called when window manager wants client to redraw part of a virtual window.

tsaWMCapabilities_t

```
typedef struct {
   ptsaDefaultCapabilities_t   defaultCapabilities;
} tsaWMCapabilities_t, *ptsaWMCapabilities_t;
```

tsaWMInstanceSetup_t

```
typedef struct {
   ptmAvPacket_t          pBackPlane;
   tsaWMStackingOrder_t   stackingOrder;
} tsaWMInstanceSetup_t, *ptsaWMInstanceSetup_t ;
```

Fields

`pBackplane`	Packet that holds the background as displayed and to which the window manager will draw. Only the packet of the instance that is set up first is used. Note: The upper left pixel in this back plane will be the background color used to redraw the background.
`stackingOrder`	Position of windows of this instance relative to other instance's windows.

Description

Sets up an instance.

Window Manager API Functions

This section presents the Window Manager API function descriptions.

Name	Page
tsaWMGetCapabilities	292
tsaWMOpen	292
tsaWMClose	293
tsaWMInstanceSetup	293
tsaWMCreateRealWindow	294
tsaWMCreateVirtualWindow	295
tsaWMDestroyWindow	296
tsaWMMoveWindow	297
tsaWMRaiseWindow	298
tsaWMLowerWindow	299
tsaWMRaiseAllWindows	300
tsaWMLowerAllWindows	300
tsaWMDisplayWindow	301
tsaWMHideWindow	302
tsaWMRedrawWindow	303
tsaWMChangeViewingWindow	304
tsaWMLockWindow	305
tsaWMUnlockWindow	306

tsaWMGetCapabilities

```
tmLibappErr_t tsaWMGetCapabilities(
   ptsaWMCapabilities_t   *ppCap
);
```

Parameters

ppCap	Pointer to a variable in which to return a pointer to the capabilities data.

Return Codes

TMLIBAPP_OK	Success.

Description

Gets the capabilities of the window manager.

tsaWMOpen

```
tmLibdevErr_t tsaWMOpen(
   Int   *pInstance
);
```

Parameters

pInstance	Pointer to the (returned) instance, to be used for other WM calls.

Return Codes

TMLIBAPP_OK	Success.

Description

Opens an instance to use the window manager.

tsaWMClose

```
tmLibappErr_t tsaWMClose(
   Int   instance
);
```

Parameters

instance	The instance to close.

Return Codes

TMLIBAPP_OK	Success.

Description

Closes an instance.

tsaWMInstanceSetup

```
tmLibappErr_t tsaWMInstanceSetup(
   Int                     instance,
   ptsaWMInstanceSetup_t   pSetup
);
```

Parameters

instance	The instance.
pSetup	Set up information.

Return Codes

TMLIBAPP_OK	Success.

Description

Sets up an instance.

tsaWMCreateRealWindow

```
tmLibappErr_t tsaWMCreateRealWindow(
   Int                    instance,
   UInt32                 parentID,
   tsaWMStackingOrder_t   stackingOrder,
   Int                    x,
   Int                    y,
   Bool                   transparent,
   Int                    transValue,
   ptmAvPacket_t          pPacket,
   UInt32                *pWindowID
);
```

Parameters

`instance`	Valid instance as returned by tsaWMOpen.
`parentID`	The ID of the parent window, 0 for 'no parent'.
`stackingOrder`	Position of the window in the stack of windows of this instance.
`x,y`	Initial position of window.
`transparent`	Whether window is transparent.
`transValue`	Value in buffer for transparent.
`pPacket`	Packet that contains the actual picture and information like size, etc. Its format must be compatible with the back plane setup in **tsaWMInstanceSetup**.
`pWindowID`	The returned ID of the created window.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Creates a real window (but does not yet display it).

tsaWMCreateVirtualWindow

```
tmLibappErr_t tsaWMCreateVirtualWindow(
   Int                        instance,
   UInt32                     parentID,
   tsaWMStackingOrder_t       stackingOrder,
   Int                        x,
   Int                        y,
   Int                        w,
   Int                        h,
   Bool                       transparent,
   ptsa2DContext_t            pGrContext,
   ptsaRedrawCallbackFun_t    pRedrawCbFun,
   UInt32                    *pWindowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**.
`parentID`	The ID of the parent window, 0 for 'no parent'.
`stackingOrder`	Position of the window in the stack of windows of this instance.
`x,y`	Initial position of window.
`w,h`	Initial width and height of window.
`transparent`	Whether window is transparent.
`pGrContext`	Graphical context used in this window.
`pRedrawCbFun`	Callback function used when the window manager wants the application to update the window.
`pWindowID`	The returned ID of the created window.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Creates a virtual window (but does not yet display it).

tsaWMDestroyWindow

```
tmLibappErr_t tsaWMDestroyWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to destroy.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Destroys a window.

tsaWMMoveWindow

```
tmLibappErr_t tsaWMMoveWindow(
   Int      instance,
   Int      x,
   Int      y,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen,** owner of the manipulated window.
`x,y`	New position of window.
`windowID`	The ID of the window to move.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Moves a window, and redraws it if any part of the window has become uncovered.

tsaWMRaiseWindow

```
tmLibappErr_t tsaWMRaiseWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to raise.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Raises a window higher than other windows of the same instance with stacking order **DONT_CARE**, and redraws it if any part of the window has become uncovered.

tsaWMLowerWindow

```
tmLibappErr_t tsaWMLowerWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to lower.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Lowers a window lower other windows of the same instance with stacking order **DONT_CARE**.

tsaWMRaiseAllWindows

```
tmLibappErr_t tsaWMRaiseAllWindows(
   Int   instance
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Raises all windows of an instance higher than windows of other instances with stacking order **DONT_CARE**, and redraws those parts of windows that have become uncovered.

tsaWMLowerAllWindows

```
tmLibappErr_t tsaWMLowerAllWindows(
   Int   instance
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Lowers all windows of an instance lower than windows of other instances with stacking order **DONT_CARE**.

tsaWMDisplayWindow

```
tmLibappErr_t tsaWMDisplayWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to display.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Displays a created window. That is, the function displays those parts of the window that are not covered by other windows, by either block transferring those parts to the backplane (in case of a real window) or calling the redraw callback function (in case of a virtual window).

tsaWMHideWindow

```
tmLibappErr_t tsaWMHideWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to hide.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Hides a window, by displaying the parts of other windows or the backplane that have become visible.

tsaWMRedrawWindow

```
tmLibappErr_t tsaWMRedrawWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to redraw.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Forces a redraw of those parts of the window that are not covered by other windows. For a real window, this means copying parts of the off screen buffer. For a virtual window, the callback redraw function is called.

tsaWMChangeViewingWindow

```
tmLibappErr_t tsaWMChangeViewingWindow(
   Int       instance,
   Int       x,
   Int       y,
   Int       w,
   Int       h,
   UInt32    windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`x,y`	Offset from left upper corner of off screen buffer.
`w,h`	Width and height of viewing window. Negative for 'no change.'
`windowID`	The ID of the real window to redraw.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Changes the part of the off-screen buffer that is visible.

tsaWMLockWindow

```
tmLibappErr_t tsaWMLockWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to lock.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Locks a window so that the window manager does not try to redraw it when part become uncovered. This may be useful when the off-screen buffer contains incorrect data.

tsaWMUnlockWindow

```
tmLibappErr_t tsaWMUnlockWindow(
   Int      instance,
   UInt32   windowID
);
```

Parameters

`instance`	Valid instance as returned by **tsaWMOpen**, owner of the manipulated window.
`windowID`	The ID of the window to redraw.

Return Codes

`TMLIBAPP_OK`	Success.

Description

Unlocks a window, and redraws the window if it has been manipulated while locked.

