My Video Player

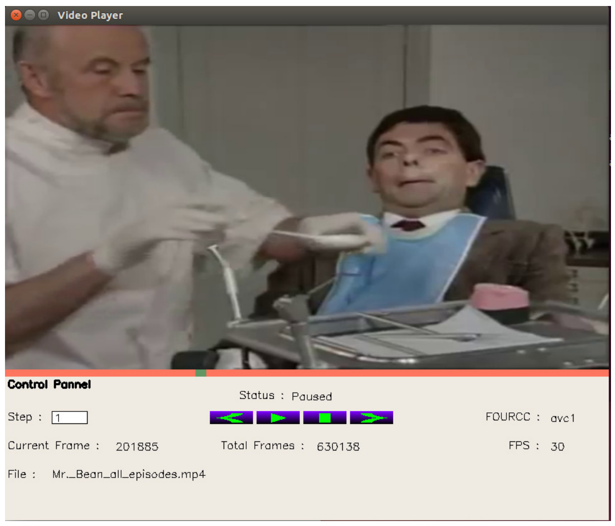
Generated by Doxygen 1.8.6

Sat Jan 3 2015 01:44:24

Contents

Chapter 1

Simple Video Player Using OpenCV



Control Pannel
Status : Paused
Step : 1
FOURCC : avc1
Current Frame : 201885
Total Frames : 630138
FPS : 30
File : Mr._Bean_all_episodes.mp4

Figure 1.1: Outline

1.1 Abstract

This document explains the code I have used to create a simple video player using the OpenCV library. One may use this as a quick guide to the most common OpenCV functions. Although I suggest this document for those who are beginners and want to quickly get accustomed to using OpenCV, I strongly recommend using the official book "Learning OpenCV", published by O'REILLY. This document will be most beneficial for those who are already familiar with image / video processing, but want to start using OpenCV library for various reasons.

Author

Milind G. Padalkar (`milind.padalkar@gmail.com`)

Date

October 2010

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 Field_Area Struct Reference

Structure to store the top-left and botton-right corner coordinates of various fields & buttons.

Data Fields

- int x1

  x coordinate of the top-left corrner.
- int y1

  y coordinate of the top-left corrner.
- int x2

  x coordinate of the bottom-right corrner.
- int y2

  y coordinate of the bottom-right corrner.

4.1.1 Detailed Description

Structure to store the top-left and botton-right corner coordinates of various fields & buttons.

At times, it is necessary to know the if the mouse is pointing to a sepcific area in the dispalyed image. Since all the buttons, text-fields, slider etc. are nothing by sub-images of the entire image, a structure is necessary to know their locations. This will enable us to call the correct callback function say, pressing the button, editing the text-box, moving the slider, etc. This structure holds the corrdinates of the top-left corner (*x1*, *y1*) and bottom-right corner (*x2*, *y2*) of the various fields.

Definition at line 152 of file video_player.c.

4.1.2 Field Documentation

4.1.2.1 int x1

x coordinate of the top-left corrner.

Definition at line 153 of file video_player.c.

4.1.2.2 int x2

x coordinate of the bottom-right corrner.

Definition at line 155 of file video_player.c.

4.1.2.3 int y1

y coordinate of the top-left corrner.

Definition at line 154 of file video_player.c.

4.1.2.4 int y2

y coordinate of the bottom-right corrner.

Definition at line 156 of file video_player.c.

The documentation for this struct was generated from the following file:

- video_player.c

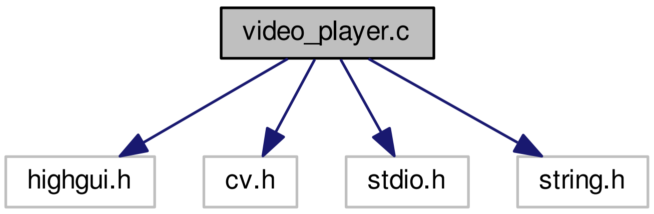
Chapter 5

File Documentation

5.1 video_player.c File Reference

```
#include <highgui.h>
#include <cv.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for video_player.c:



Data Structures

- struct Field_Area

  Structure to store the top-left and botton-right corner coordinates of various fields & buttons.

Macros

- #define sldr_btn_width 15

  Default value for the Slider Button's width.
- #define sldr_height 10

  Default value for the Slider Button's height.
- #define ctrl_pnl_height 200

  Default value for Control Pannel's height.
- #define p_width 840

  Width of the video player.

- #define scrn_height 480

    Height of the video-display area.
- #define p_height (scrn_height + sldr_height + ctrl_pnl_height)

    Height of the video player.
- #define MOUSE_CALLBACK 0

    Alias for function call made by the MOUSE's callback.
- #define OTHER_CALLS 1

    Alias for function call made by any function other than MOUSE's callback or Textbox Editor's function.
- #define EDIT_CALLS 2

    Alias for function call made by functions the edit the textboxes function. This is reserved for future.
- #define STATIC_TEXT 0

    Alias for static-text field.
- #define EDIT_TEXT 1

    Alias for text-box field.
- #define PLAY_BTN 0

    Alias for play button.
- #define PAUSE_BTN 1

    Alias for pause button.
- #define STOP_BTN 2

    Alias for stop button.
- #define STEPUP_BTN 3

    Alias for step-up button.
- #define STEPDOWN_BTN 4

    Alias for step-down button.
- #define BTN_ACTIVE 0

    Alias for an active button.
- #define BTN_INACTIVE 1

    Alias for an inactive button.

Functions

- void resetField (IplImage ∗image, int text_type)

    Function to reset a given text field.
- void initialize_pnl (char ∗filename)

    Function to initialise the control pannel.
- int moveSlider (int pos, int call_from)

    Custome slider's callback function.
- void my_mouse_callback (int event, int x, int y, int flags, void ∗param)

    Mouse's callback function.
- void getButton (IplImage ∗image, int btn_type, int btn_state)

    Function to get a new button.
- void getSpectrumVert (IplImage ∗image, CvScalar color1, CvScalar color2)

    Function to vertically color a button.
- void getSpectrumHorz (IplImage ∗image, CvScalar color1, CvScalar color2)

    Function to horizontaly color a button.
- void draw_triangle (IplImage ∗image, CvScalar color)

    Function to draw a triangle on a given image.
- void draw_square (IplImage ∗image, CvScalar color)

    Function to draw a square on a given image.
- void draw_pause (IplImage ∗image, CvScalar color)

Function to draw a pause symbol on a given image.
- void draw_stepup (IplImage ∗image, CvScalar color)

  Function to draw a step-up symbol on a given image.
- void draw_stepdown (IplImage ∗image, CvScalar color)

  Function to draw a step-down symbol on a given image.
- void fill_color (IplImage ∗image, CvScalar color)

  Function to fill a symbol with a given color.
- void change_status ()

  Function to change the status message.
- void type_step (char c, int frame_val)

  Function to edit a textbox.
- void resetAllEdits ()

  Function to reset all fields to their previous contents.
- int main (int argc, char ∗∗argv)

Variables

- CvCapture ∗ vid

  Pointer to CvCapture structure.
- IplImage ∗ player

  Pointer to the main image.
- IplImage ∗ pnl

  Pointer to the control-pannel sub-image.
- IplImage ∗ slider

  Pointer to the slider-strip sub-image.
- IplImage ∗ sldr_btn

  Pointer to the slider-button sub-image.
- IplImage ∗ sldr_val

  Pointer to the slider-value static-text sub-image.
- IplImage ∗ oslider

  Pointer to temporary slider-value static-text sub-image.
- IplImage ∗ frame_area

  Pointer to the frame-area sub-image.
- IplImage ∗ frame

  Pointer to the fetched frame sub-image.
- IplImage ∗ old_frame

  Pointer to the previously fetched frame.
- IplImage ∗ cur_frame_no

  Pointer to current frame number static-text.
- IplImage ∗ fps_edit

  Pointer to FPS (Frames Per Second) static-text.
- IplImage ∗ four_cc_edit

  Pointer to FOUR_CC static-text.
- IplImage ∗ status_edit

  Pointer to "Status" static-text.
- IplImage ∗ numFrames

  Pointer to Total Frames static-text.
- IplImage ∗ step_edit

  Pointer to the Step textbox.
- IplImage ∗ play_pause_btn

Pointer to play/pause button area.
- IplImage ∗ stop_btn

  Pointer to stop button area.
- IplImage ∗ stepup_btn

  Pointer to step_up button area.
- IplImage ∗ stepdown_btn

  Pointer to step_down button area.
- int sldr_start

  Indicates the starting position (frame number) of the slider.
- int sldr_maxval

  The maximum number of frames in the video.
- int step_val = 1

  Step size.
- char line [20]

  Memory to hold any string temporarily.
- char edit_text [20]

  Memory to hold a textbox string temporarily.
- char status_line [15]

  Memory to hold the "status" string.
- char four_cc_str [4]

  Memory to hold the Four Character Code (FOUR_CC).
- double fps

  Frames per second.
- long fourcc_l

  Four Character Code.
- char ∗ fourcc

  Four_CC temporary string.
- int blink_count = 0

  Blinker count.
- int blink_max = 5
- char blink_char = '|'

  Threshold to toogle the blink_char.
- Field_Area play_pause_btn_area

  The blinking character, toogled with an underscore (_).
- Field_Area stop_btn_area

  Stop Button coordinates.
- Field_Area stepup_btn_area

  Step Up Button coordinates.
- Field_Area stepdown_btn_area

  Step Down Button coordinates.
- Field_Area fps_edit_area

  FPS static-text coordinates.
- Field_Area four_cc_edit_area

  FOUR_CC static-text coordinates.
- Field_Area status_edit_area

  Status string coordinates.
- Field_Area step_edit_area

  Step textbox coordinates.
- bool sldr_moving = false

  Ture when slider is moving.
- bool playing = false

True when the video is being played.

- bool processing = false

  True when some processing is carried out.
- bool typing_step = false

  True when any textbox value is being edited.
- bool blinking = false

  True when blinking character is set.
- CvScalar red = cvScalar(0, 0, 255)

  Red color.
- CvScalar green = cvScalar(0, 255, 0)

  Green color.
- CvScalar blue = cvScalar(255, 0, 0)

  Blue color.
- CvScalar black = cvScalar(0, 0, 0)

  Black color.
- CvScalar white = cvScalar(255, 255, 255)

  White color.
- CvScalar light_yellow = cvScalar(242, 255, 255)

  Light Yellow color.
- CvScalar yellow = cvScalar(0, 255, 255)

  Yellow color.
- CvScalar gray = cvScalar(242, 242, 242)

  Gray color.
- CvScalar orange = cvScalar(0, 242, 255)

  Orange color.
- CvScalar voilet = cvScalar(255, 0, 127)

  Voilet color.
- CvScalar brown = cvScalar(0, 0, 127)

  Brown color.
- CvFont font

  Normal font.
- CvFont font_italic

  Italic font.
- CvFont font_bold

  Bold font.
- CvFont font_bold_italic

  Bold Italic font.
- int font_face_italic = CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC

  Font face.
- int font_face = CV_FONT_HERSHEY_SIMPLEX

  Font face.
- double hscale = 0.5

  Font's Horizontal Scale parameter.
- double vscale = 0.5

  Font's Vertical Scale parameter.
- double shear = 0

  Font's Shear parameter.
- int thickness = 1

  Font's Thickness parameter.
- int line_type = 8

  Font's Line-type parameter.

5.1.1 Detailed Description

File containing the source code of this simple video player.

Definition in file video_player.c.

5.1.2 Macro Definition Documentation

5.1.2.1 #define BTN_ACTIVE 0

Alias for an *active* button.

If this value is passed, then the button under consideration is active. Therefore, all the operations on the pressing the button will be possible.

Definition at line 140 of file video_player.c.

5.1.2.2 #define BTN_INACTIVE 1

Alias for an *inactive* button.

If this value is passed, then the button under consideration is inactive. Therefore, no operations will be possible on pressing this button i.e. the button's callback function will not be called. Currently, no buttons are inactive during any point of execution. This is reserved for future enhancements in the video player.

Definition at line 146 of file video_player.c.

5.1.2.3 #define ctrl_pnl_height 200

Default value for Control Pannel's height.

The control pannel is nothing but a sub-image. This value specifies the height of this sub-image. The width is same as that of the player (the main image displayed on the screen) width.

Definition at line 49 of file video_player.c.

5.1.2.4 #define EDIT_CALLS 2

Alias for function call made by functions *the edit the textboxes function.* This is reserved for future.

If this value is passed, then the function call is made by the functions editing the textboxes (for future use). Curretly, this value is meaningless.

Definition at line 89 of file video_player.c.

5.1.2.5 #define EDIT_TEXT 1

Alias for *text-box* field.

If this value is passed, then the text field under consideration is a text-box. Accordingly operations are to be carried out on this text field.

Definition at line 102 of file video_player.c.

5.1.2.6 #define MOUSE_CALLBACK 0

Alias for *function call made by the MOUSE's callback.*

If this value is passed, then the function call is made by the MOUSE'S callback function. Sometimes the information about the caller function is required. This alias is easy to remember & is therefore associated to the MOUSE's callback function.

Definition at line 77 of file video_player.c.

5.1.2.7 #define OTHER_CALLS 1

Alias for function call made by any function other than *MOUSE's callback or Textbox Editor's function.*

If this value is passed, then the function call is made by any function other than the MOUSE'S callback function or functions editing the textboxes (for future use). Curretly, this value specifies that the call is made from any function other than the MOUSE's callback function.

Definition at line 83 of file video_player.c.

5.1.2.8 #define p_height (scrn_height + sldr_height + ctrl_pnl_height)

Height of the video player.

This value defines the height of the video player i.e. the main image. This height is the addition of the heights of *display area*, the *slider height* and the *height of the control pannel.*

See Also

p_width

Definition at line 69 of file video_player.c.

5.1.2.9 #define p_width 840

Width of the video player.

This value defines the width of the main image (player) displayed on the screen. Various areas like the area of the video being displayed, the different textboxes, etc are actually sub-images of this main image.

See Also

p_height

Definition at line 56 of file video_player.c.

5.1.2.10 #define PAUSE_BTN 1

Alias for *pause* button.

If this value is passed, then the button under consideration is pause-button. Accordingly operations are to be carried out on the button area.

Definition at line 115 of file video_player.c.

5.1.2.11 #define PLAY_BTN 0

Alias for *play* button.

If this value is passed, then the button under consideration is play-button. Accordingly operations are to be carried out on the button area.

Definition at line 109 of file video_player.c.

5.1.2.12 #define scrn_height 480

Height of the video-display area.

This value defines the height of the video-display area. This is the area where the actual video frame is displayed. For convinence, every video frame is scaled to $p\_width \times scrn\_height$ before being displayed.

Definition at line 62 of file video_player.c.

5.1.2.13 #define sldr_btn_width 15

Default value for the Slider Button's width.

The slider button's width is set using this value.

Definition at line 37 of file video_player.c.

5.1.2.14 #define sldr_height 10

Default value for the Slider Button's height.

The slider button's height is set using this value.

Definition at line 43 of file video_player.c.

5.1.2.15 #define STATIC_TEXT 0

Alias for *static-text* field.

If this value is passed, then the text field under consideration is static-text. Accordingly operations are to be carried out on this text field.

Definition at line 96 of file video_player.c.

5.1.2.16 #define STEPDOWN_BTN 4

Alias for *step-down* button.

If this value is passed, then the button under consideration is step-down button. Accordingly operations are to be carried out on the button area.

Definition at line 133 of file video_player.c.

5.1.2.17 #define STEPUP_BTN 3

Alias for *step-up* button.

If this value is passed, then the button under consideration is step-up button. Accordingly operations are to be carried out on the button area.

Definition at line 127 of file video_player.c.

5.1.2.18 #define STOP_BTN 2

Alias for *stop* button.

If this value is passed, then the button under consideration is stop-button. Accordingly operations are to be carried out on the button area.

Definition at line 121 of file video_player.c.

5.1.3 Function Documentation

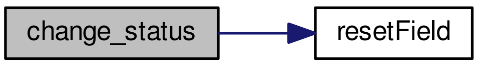
5.1.3.1 void change_status ( )

Function to change the status message.

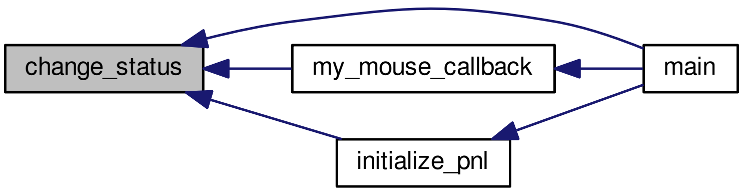
Definition at line 1405 of file video_player.c.

```
                        {
    resetField( status_edit, STATIC_TEXT );
    cvPutText( status_edit, status_line, cvPoint( 3,
  status_edit->height - 8 ), &font, black );
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.2 void draw_pause (IplImage ∗ *image*, CvScalar *color*)

Function to draw a pause symbol on a given image.

Function to draw a two parallel rectangles for the pause button. We pass the sub-image where we want to create the pause button and also pass the color which we desire of the button. We first define 2 points for the first rectangle whose coordinates are stored in pt1 and pt2 and a line between these points would be a vertical line. Now we simply draw 5 lines parallel to this line for the first rectangle and also 5 parallel lines for the second rectangle

Parameters

image	: The image where we want to place the pause-rectangles

color	: The desired color

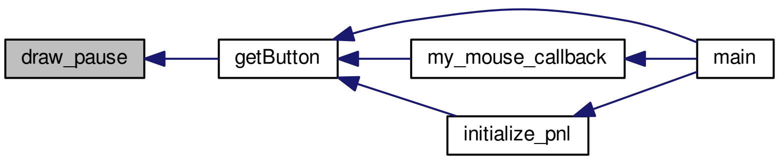
See Also

cvPoint(), **cvLine()**.

Definition at line 1305 of file video_player.c.

```
                                                   {
         int y_start = 3;
         int y_end = image->height - y_start;
         int dist = 3;
         CvPoint pt1, pt2, tmp1, tmp2;
         pt1.x = image->width/2;
         pt1.y = y_start;
         pt2.x = pt1.x;
         pt2.y = y_end;
         for( int col=0; col<5; col++  ){
             tmp1.x = pt1.x + dist + col;
             tmp1.y = pt1.y;
             tmp2.x = pt2.x + dist + col;
             tmp2.y = pt2.y;
             cvLine( image, tmp1, tmp2, color );
             tmp1.x = pt1.x - dist - col;
             tmp1.y = pt1.y;
             tmp2.x = pt2.x - dist - col;
             tmp2.y = pt2.y;
             cvLine( image, tmp1, tmp2, color );
         }
     }
```

Here is the caller graph for this function:



5.1.3.3 void draw_square (IplImage ∗ *image*, CvScalar *color*)

Function to draw a square on a given image.

Function to draw a square for the stop button. We pass the sub-image where we want to create the stop button and also pass the color which we desire of the button. We first define 4 points for the square whose coordinates are stored in pt1, pt2, pt3 and pt4. Now we simply draw segments to connect these points and finally fill up the square with the desired color.

Parameters

image	: The image where we want to place the stop-square
color	: The desired color

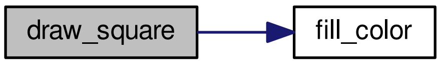
See Also

cvPoint(), **cvRectangle**, fill_color.

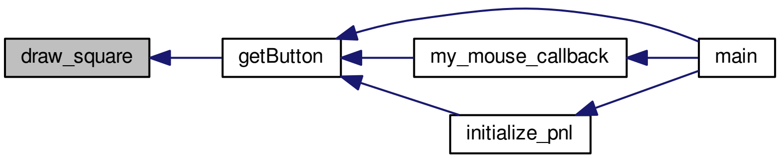
Definition at line 1285 of file video_player.c.

```
                                                    {
    CvPoint pt1, pt2;
    pt1.x = 3*image->width/8;
    pt1.y = 3;
    pt2.x = 5*image->width/8;
    pt2.y = image->height - pt1.y;
    cvRectangle( image, pt1, pt2, color );
    fill_color( image, color );
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



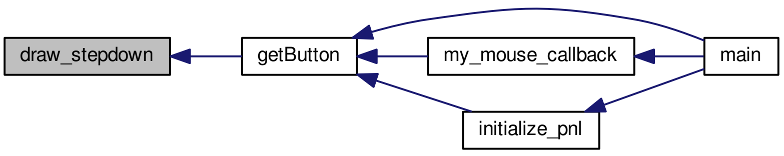
5.1.3.4 void draw_stepdown (IplImage ∗ *image,* CvScalar *color*)

Function to draw a step-down symbol on a given image.

Definition at line 1382 of file video_player.c.

```
                                                      {
    CvPoint pt1, pt2, pt3, pt4;
    pt1.x = 4*image->width/8;
    pt2.x = 6*image->width/8;
    pt3.x = pt1.x;
    pt4.x = pt2.x;
    int y_start = 3;
    int y_end = image->height/2 ;
    for( int row=y_start; row<=y_end; row++ ){
        pt1.x = pt1.x - row + y_start;
        pt2.x = pt2.x - row + y_start;
        pt1.y = row;
        pt2.y = row;
        pt3.x = pt1.x;
        pt4.x = pt2.x;
        pt3.y = image->height - row;
        pt4.y = pt3.y;
        cvLine( image, pt1, pt2, color );
        cvLine( image, pt3, pt4, color );
    }
}
```

Here is the caller graph for this function:



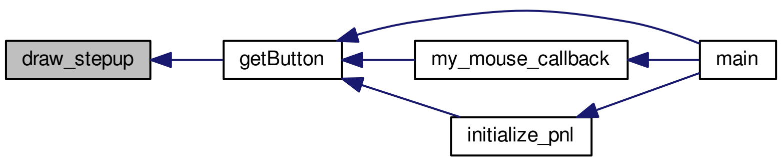
5.1.3.5 void draw_stepup (IplImage ∗ *image,* CvScalar *color*)

Function to draw a step-up symbol on a given image.

Definition at line 1359 of file video_player.c.

```
                                                     {
         CvPoint pt1, pt2, pt3, pt4;
         pt1.x = 2*image->width/8;
         pt2.x = 4*image->width/8;
         pt3.x = pt1.x;
         pt4.x = pt2.x;
         int y_start = 3;
         int y_end = image->height/2 ;
         for( int row=y_start; row<=y_end; row++ ){
             pt1.x = pt1.x + row - y_start;
             pt2.x = pt2.x + row - y_start;
             pt1.y = row;
             pt2.y = row;
             pt3.x = pt1.x;
             pt4.x = pt2.x;
             pt3.y = image->height - row;
             pt4.y = pt3.y;
             cvLine( image, pt1, pt2, color );
             cvLine( image, pt3, pt4, color );
         }
 }
```

Here is the caller graph for this function:



5.1.3.6 void draw_triangle (IplImage ∗ *image,* CvScalar *color*)

Function to draw a triangle on a given image.

Function to draw a triangle for the play button. We pass the sub-image where we want to create the play button and also pass the color which we desire of the button. We first define 3 points for the triangle whose coordinates are stored in pt1, pt2 and pt3. Now we simply draw segments to connect these points and finally fill up the triangle with the desired color.

Parameters

image	: The image where we want to place the play-triangle
color	: The desired color

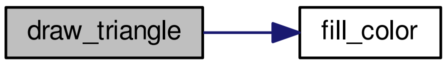
See Also

cvPoint(), cvLine(), fill_color.

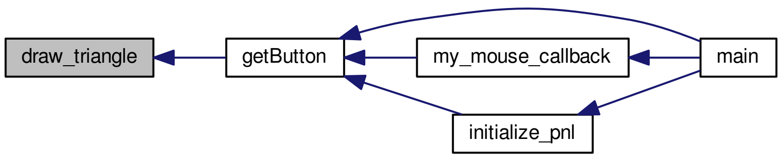
Definition at line 1261 of file video_player.c.

```
                                                    {
    CvPoint pt1, pt2, pt3;
    pt1.x = image->width/3;
    pt1.y = 3;
    pt2.x = pt1.x;
    pt2.y = image->height - pt1.y;
    pt3.x = 2*pt1.x;
    pt3.y = image->height/2;
    cvLine( image, pt1, pt2, color );
    cvLine( image, pt3, pt2, color );
    cvLine( image, pt1, pt3, color );
    fill_color( image, color );
}
```

Here is the call graph for this function:


$$\begin{tikzcd}\boxed{\text{draw\_triangle}} \arrow[r] & \boxed{\text{fill\_color}}\end{tikzcd}$$

Here is the caller graph for this function:


$$\begin{tikzcd}\boxed{\text{draw\_triangle}} & \boxed{\text{getButton}} \arrow[l] & \boxed{\text{my\_mouse\_callback}} \arrow[l] & \boxed{\text{main}} \arrow[l] \arrow[ll, bend right] \arrow[dl] \\ & & \boxed{\text{initialize\_pnl}} \arrow[ul] & \end{tikzcd}$$

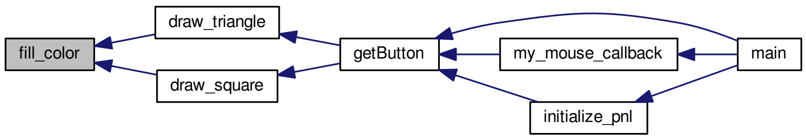
5.1.3.7 void fill_color (IplImage * *image*, CvScalar *color*)

Function to fill a symbol with a given color.

Definition at line 1331 of file video_player.c.

```
                                                 {
    bool start_fill = false;
    for( int row=0; row<image->height; row++ ){
        uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
        for( int col=0; col<image->width; col++ ){
            if(
                ( ptr[ col*image->nChannels + 0 ] == color.val[0] ) &&
                ( ptr[ col*image->nChannels + 1 ] == color.val[1] ) &&
                ( ptr[ col*image->nChannels + 2 ] == color.val[2] )
            ){
                if( !start_fill ){
                    start_fill = true;
                }
                else{
                    start_fill = false;
                    break;
                }
            }
            if( start_fill ){
                ptr[ col*image->nChannels + 0 ] = color.val[0];
                ptr[ col*image->nChannels + 1 ] = color.val[1];
                ptr[ col*image->nChannels + 2 ] = color.val[2];
            }
        }
    }
}
```

Here is the caller graph for this function:



5.1.3.8 void getButton (IplImage ∗ *image,* int *btn_type,* int *btn_state*)

Function to get a new button.

Function to get the desired control button, say play, pause, stop, stepup, stepdown. The buttons are nothing but sub-images.

Parameters

image	: This is the sub-image for the desired button.
btn_type	: Can be any of the following viz. PLAY_BTN, PAUSE_BTN, STOP_BTN, STEPUP_BTN, STEPDOWN_BTN.
btn_state	: Can be either BTN_ACTIVE or BTN_INACTIVE. For the time being, only BTN_ACTIVE is used and it is meaningless to pass BTN_INACTIVE.

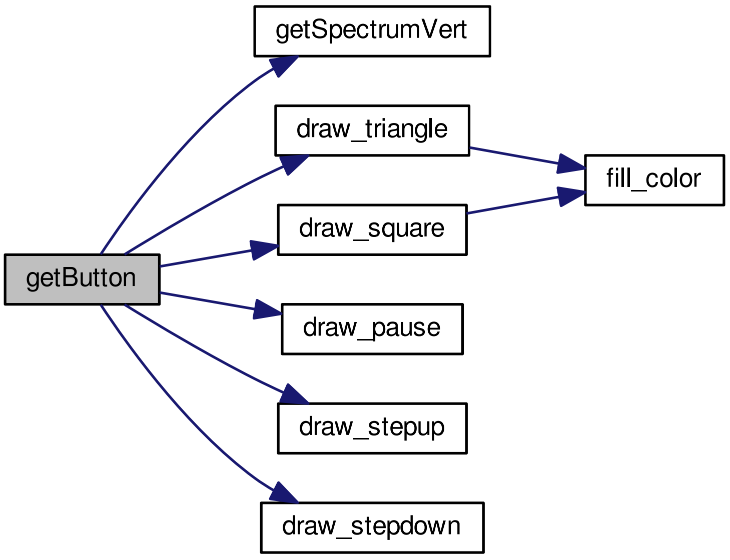
See Also

IplImage

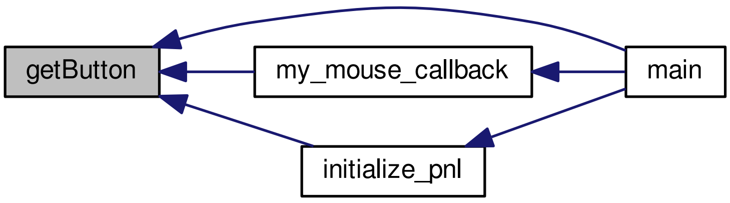
Definition at line 991 of file video_player.c.

```
                                                         {
    getSpectrumVert( image, voilet, black );
    if( btn_type==PLAY_BTN ){
        draw_triangle( image, green );
    }
    if( btn_type==STOP_BTN ){
        draw_square( image, green );
    }
    if( btn_type==PAUSE_BTN ){
        draw_pause( image, green );
    }
    if( btn_type==STEPUP_BTN ){
        draw_stepup( image, green );
    }
    if( btn_type==STEPDOWN_BTN ){
        draw_stepdown( image, green );
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.9 void getSpectrumHorz (IplImage ∗ *image*, CvScalar *color1*, CvScalar *color2*)

Function to horizontaly color a button.

This function is nothing by an implementation of linear interploation horizontally i.e. The *color1* is the color of the leftmost row of the sub-image (*image*) and *color2* is the color of the rightmost row of the sub-image. The intermediate colors are calculated using the following formula.

$$X = \frac{B-A}{L} \times l + A$$

where,

X : color of the pixel to be determined.

B : Color of the rightmost row.

A : Color of the leftmost row.

l : Distance of the current pixel from the left (i.e. column number).

L : Total number of columns.

Parameters

image	: The input sub-image for the button to be colored.
color1	: Color of the leftmost row.
color2	: Color of the rightmost row.

See Also

`IplImage`

Definition at line 1084 of file video_player.c.

```
                                                                          {
    //Color the leftmost and rightmost pixels of each row with with color1 and color2 respectively
    for( int row=0; row<image->height; row++ ){
        uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
        for( int chl=0; chl<image->nChannels; chl++ ){
            ptr[chl] = color1.val[chl];
            ptr[ ( image->width-1 )*image->nChannels + chl ] = color2.val[ chl ];
        }
    }
    //Interploation applied here
    //b_a_L => (B-A)/L... (X-A)/l = (B-A)/L :: => X = (((B-A)/L)*l + A)
    for( int row=0; row<image->height; row++ ){
        uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
        for( int col=0; col<image->width; col++ ){
            for( int chl=0; chl<image->nChannels; chl++ ){
                ptr[ col*image->nChannels + chl  ] =  ( ptr[ ( image->width - 1 )*image->nChannels + chl ]
 - ptr[ chl ] )*( col/( float )image->width ) + ptr[ chl ];
            }
        }
    }
}
```

5.1.3.10 void getSpectrumVert (IplImage * *image*, CvScalar *color1*, CvScalar *color2*)

Function to vertically color a button.

This function is nothing by an implementation of linear interploation vertically i.e. The *color1* is the color of the topmost row of the sub-image (*image*) and *color2* is the color of the bottom most row of the sub-image. The intermediate colors are calculated using the following formula.

$$X = \frac{B-A}{L} \times l + A$$

where,

X : color of the pixel to be determined.

B : Color of the bottommost row.

A : Color of the topmost row.

l : Distance of the current pixel from the top (i.e. row number).

L : Total number of rows.

Parameters

image	: The input sub-image for the button to be colored.
color1	: Color of the topmost row.
color2	: Color of the bottommost row.

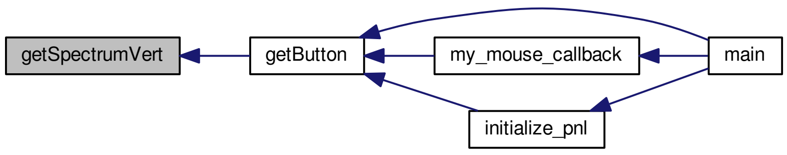
See Also

IplImage

Definition at line 1029 of file video_player.c.

```
                                                                          {
    for( int row=0; row<image->height; row++ ){
        //If topmost row is selected
        if( row==0 ){
            uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
            for( int col=0; col<image->width; col++ ){
                for( int chl=0; chl<image->nChannels; chl++ ){
                    ptr[ col*image->nChannels + chl] = color1.val[chl];
                }
            }
        }
        //If bottommost row is selected
        if( row==image->height-1 ){
            uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
            for( int col=0; col<image->width; col++ ){
                for( int chl=0; chl<image->nChannels; chl++ ){
                    ptr[ col*image->nChannels + chl] = color2.val[chl];
                }
            }
        }
    }

    //Interpolation is applied
    //b_a_L => (B-A)/L... (X-A)/l = (B-A)/L :: => X = (((B-A)/L)*l + A)
    for( int row=1; row<image->height-1; row++ ){
        uchar *ptr = ( uchar* )( image->imageData );
        for( int col=0; col<image->width; col++ ){
            for( int chl=0; chl<image->nChannels; chl++ ){
                ptr[ row*image->widthStep + col*image->nChannels + chl  ] = (
                    ptr[ ( image->height-1 )*image->widthStep + col*image->nChannels + chl ] - ptr[ col*
image->nChannels + chl ] )*(
                        row/( float )image->height ) + ptr[ col*image->nChannels + chl ];
            }
        }
    }
}
```

Here is the caller graph for this function:



5.1.3.11 void initialize_pnl (char * *filename*)

Function to initialise the control pannel.

This function is for initializing the control pannel, adding textfields, text and buttons to it. One may learn how to create a sub-image from an existing IplImage. Each sub-image can act as an independent image. The advantage of using sub-image over ROI is that multiple parts of an image can be worked upon simultaneously. You will come to know the importance of sub-image in the following where you will see how text-boxes and buttons are created with the use of sub-images. Lets's start.

In the initial lines you will come across the `cvPutText()` function. This is used to place text at various parts of the control pannel sub-image. Next you come across are the *row* and *col*. These are the starting coordinates where you want to place the textfields or buttons. To create such fields and buttons (which are nothing but sub-images of the control pannel, which is again a sub-image of the video-player), declare the IplImage header using `cvCreateImageHeader()` using the required dimensions. This will only create the image header and no data is assigned to it. To use this as a sub-image we need to need to make the following assignments.

1. sub_image->origin = parent_image->origin.
2. sub_image->widthStep = parent_image->widthStep.
3. sub_image->imageData = parent_image->imageData + row*sub_image->widthStep + col*sub_image->nChannels.

Now the sub-image can be used as if it were an independent image. Further, for a few operations we need to keep a track of the coordinates of this newly created image. We store them in Field_Area structure of the respective button or text-field using the naming convention as *sub_image_area*. If the sub-image is a text-field then resetField() function is called. If the sub-image is a button then getButton() function is called.

Parameters

filename	: The absolute path of the video file to be played.

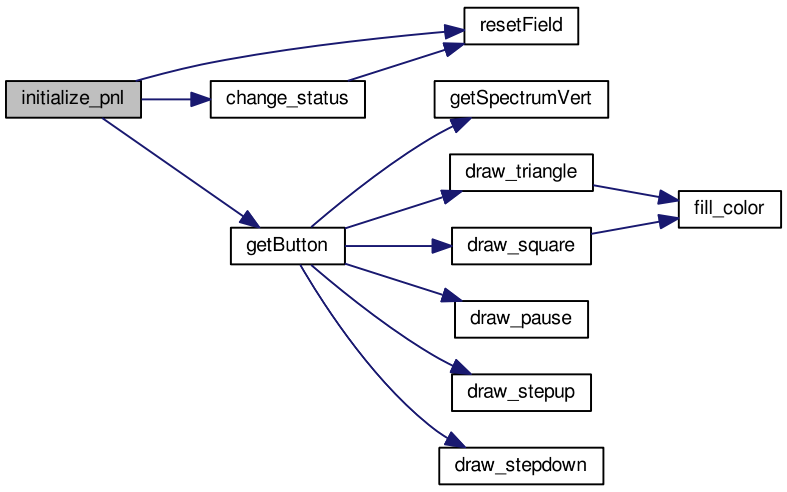
See Also

`cvPutText()`, `cvPoint()`, `cvCreateImageHeader()`, resetField(), getButton().

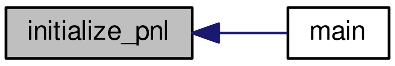
Definition at line 1122 of file video_player.c.

```
                                     {
    int row, col;
    cvPutText( pnl, "Step : ", cvPoint( 3, 60 ), &font, black );
    cvPutText( pnl, "File : ", cvPoint( 3, 140 ), &font, black );
    cvPutText( pnl, filename, cvPoint( 65, 140 ), &font, black );
    cvPutText( pnl, "Control Pannel", cvPoint( 3, 15 ), &font_bold_italic,
 black );
    cvPutText( pnl, "FPS : ", cvPoint( 700, 100 ), &font, black );
    cvPutText( pnl, "Current Frame : ", cvPoint( 3, 100 ), &font, black );
    cvPutText( pnl, "Total Frames : ", cvPoint( 300, 100 ), &font, black );
    cvPutText( pnl, "FOURCC : ", cvPoint( 668, 60 ), &font, black );
    cvPutText( pnl, "Status : ", cvPoint( 325, 30 ), &font, black );
    //Current Frame field
    row = 88;
    col = 150;
    cur_frame_no = cvCreateImageHeader( cvSize( 120, 18), IPL_DEPTH_8U, 3 );
    cur_frame_no->origin = pnl->origin;
    cur_frame_no->widthStep = pnl->widthStep;
    cur_frame_no->imageData = pnl->imageData + row*pnl->widthStep + col*
 pnl->nChannels;
    resetField( cur_frame_no, STATIC_TEXT );
    //number of frames field
    row = 88;
    col = 430;
    numFrames = cvCreateImageHeader( cvSize( 120, 18), IPL_DEPTH_8U, 3 );
    numFrames->origin = pnl->origin;
    numFrames->widthStep = pnl->widthStep;
    numFrames->imageData = pnl->imageData + row*pnl->widthStep + col*
 pnl->nChannels;
    resetField( numFrames, STATIC_TEXT );
    //Step field
    row = 48;
    col = 65;
    step_edit = cvCreateImageHeader( cvSize( 50, 18), IPL_DEPTH_8U, 3 );
    step_edit->origin = pnl->origin;
    step_edit->widthStep = pnl->widthStep;
    step_edit->imageData = pnl->imageData + row*pnl->widthStep + col*
 pnl->nChannels;
    resetField( step_edit, EDIT_TEXT );
    step_edit_area.x1 = col;
    step_edit_area.x2 = col + step_edit->width;
    step_edit_area.y1 = p_height - ctrl_pnl_height + row;
    step_edit_area.y2 = p_height - ctrl_pnl_height + step_edit->height + row;
    sprintf( line, "%d", step_val );
    cvPutText( step_edit, line, cvPoint( 3, step_edit->height - 4 ), &font, black );
    //FPS field
    row = 88;
    col = 755;
    fps_edit = cvCreateImageHeader( cvSize( 50, 18), IPL_DEPTH_8U, 3 );
    fps_edit->origin = pnl->origin;
    fps_edit->widthStep = pnl->widthStep;
    fps_edit->imageData = pnl->imageData + row*pnl->widthStep + col*pnl->nChannels;
    resetField( fps_edit, STATIC_TEXT );
    fps_edit_area.x1 = col;
    fps_edit_area.x2 = col + fps_edit->width;
    fps_edit_area.y1 = p_height - ctrl_pnl_height + row;
    fps_edit_area.y2 = p_height - ctrl_pnl_height + fps_edit->height + row;
    //FOURCC field
    row = 48;
    col = 755;
    four_cc_edit = cvCreateImageHeader( cvSize( 50, 22), IPL_DEPTH_8U, 3 );
    four_cc_edit->origin = pnl->origin;
    four_cc_edit->widthStep = pnl->widthStep;
    four_cc_edit->imageData = pnl->imageData + row*pnl->widthStep + col*pnl->nChannels;
    resetField( four_cc_edit, STATIC_TEXT );
    four_cc_edit_area.x1 = col;
    four_cc_edit_area.x2 = col + four_cc_edit->width;
    four_cc_edit_area.y1 = p_height - ctrl_pnl_height + row;
    four_cc_edit_area.y2 = p_height - ctrl_pnl_height + four_cc_edit->height + row;
    //Play/Pause button
    row = 48;
    col = 350;
    play_pause_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
    play_pause_btn->origin = pnl->origin;
    play_pause_btn->widthStep = pnl->widthStep;
    play_pause_btn->imageData = pnl->imageData + row*pnl->widthStep + col*pnl->nChannels;
    getButton( play_pause_btn, PLAY_BTN, BTN_ACTIVE );
    play_pause_btn_area.x1 = col;
    play_pause_btn_area.x2 = col + play_pause_btn->width;
    play_pause_btn_area.y1 = p_height - ctrl_pnl_height + row;
    play_pause_btn_area.y2 = p_height - ctrl_pnl_height + play_pause_btn->height + row;
    //Stop button
    row = 48;
    col = 415;
    stop_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
    stop_btn->origin = pnl->origin;
    stop_btn->widthStep = pnl->widthStep;
    stop_btn->imageData = pnl->imageData + row*pnl->widthStep + col*pnl->nChannels;
    getButton( stop_btn, STOP_BTN, BTN_ACTIVE );
    stop_btn_area.x1 = col;
    stop_btn_area.x2 = col + stop_btn->width;
    stop_btn_area.y1 = p_height - ctrl_pnl_height + row;
    stop_btn_area.y2 = p_height - ctrl_pnl_height + stop_btn->height + row;
    //Stepup button
    row = 48;
    col = 480;
    stepup_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
    stepup_btn->origin = pnl->origin;
    stepup_btn->widthStep = pnl->widthStep;
    stepup_btn->imageData = pnl->imageData + row*pnl->widthStep + col*pnl->nChannels;
    getButton( stepup_btn, STEPUP_BTN, BTN_ACTIVE );
    stepup_btn_area.x1 = col;
    stepup_btn_area.x2 = col + stepup_btn->width;
    stepup_btn_area.y1 = p_height - ctrl_pnl_height + row;
    stepup_btn_area.y2 = p_height - ctrl_pnl_height + stepup_btn->height + row;
    //Stepdown button
    row = 48;
    col = 285;
    stepdown_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
    stepdown_btn->origin = pnl->origin;
    stepdown_btn->widthStep = pnl->widthStep;
    stepdown_btn->imageData = pnl->imageData + row*pnl->widthStep + col*pnl->nChannels;
    getButton( stepdown_btn, STEPDOWN_BTN, BTN_ACTIVE );
    stepdown_btn_area.x1 = col;
    stepdown_btn_area.x2 = col + stepdown_btn->width;
    stepdown_btn_area.y1 = p_height - ctrl_pnl_height + row;
    stepdown_btn_area.y2 = p_height - ctrl_pnl_height +
 stepdown_btn->height + row;
    //Status Field
    row = 18;
    col = 395;
    status_edit = cvCreateImageHeader( cvSize( 130, 22), IPL_DEPTH_8U, 3 );
    status_edit->origin = pnl->origin;
    status_edit->widthStep = pnl->widthStep;
    status_edit->imageData = pnl->imageData + row*pnl->widthStep + col*
 pnl->nChannels;
    resetField( status_edit, STATIC_TEXT );
    status_edit_area.x1 = col;
    status_edit_area.x2 = col + status_edit->width;
    status_edit_area.y1 = p_height - ctrl_pnl_height + row;
    status_edit_area.y2 = p_height - ctrl_pnl_height +
 status_edit->height + row;
    sprintf( status_line, "Stopped" );
    change_status();
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.12 int main (int *argc,* char ∗∗ *argv*)

The main function creates the main image and various sub-images that constitute the video player. Once this outline is ready, frames from the video are fetched and displayed on the sub-image denoting the "screen area".

Simultaneously, contents of other sub-image (frame number, slider position) are also updated. Before starting to initialize the various sub-images, the fonts to be used need to be initialized. The fonts are initialized using the **cvInitFont()** function.

This is followed by the creation of an empty image (which serves as the main image of the player). The *player* image is created using the various dimensions shown earlier.

Then the control-pannel sub-image is assigned as a part of the main *player* image.

All the buttons, textboxes, static-texts, etc are initialized.

Above the control-pannel, a sub-image is assigned to be a slider. OpenCV has an inbuilt function **cvCreate-Trackbar()** to create a slider. But the disadvantage with this function is, the slider is placed at either at the top or the buttom of an image in a window. Therefore, to have the slider at a custom location in the window, I created my own slider. Practically, this slider is a sub-image to which I have assigned a mouse_callback function. Setting the ROI to this sub-image was possible, but then simultaneously accessing all the sub-images would not had been possible. Therefore, the slider sub-image is created by first creating the sub-image of the required dimensions and then setting the origin, widthstep to be the same as that of the main image and the imageData to the appropriate value of imageData of the main image. Everytime the slider position is updated, the original slider needs to be restored first and then the new position is to be marked. Therefore, the original slider sub-image is cloned to *oslider* sub-image. *sldr_val* sub-image is nothing but a rectangular image at a position derived from the slider's value. Thus, every time the slider's value is updated, the original slider sub-image (*oslider*) is restored, followed by placing the *sldr_val* sub-image at it appropriate position on the slider.

The main player images needs to be displayed using a *Named Window*. Using the **cvNamedWindow()** function we create a display window.

Everytime a mouse action (move, click, etc) occurs on the main display window, the events need to be captured and appropriate actions are to be called. For achieveing this task the **cvSetMouseCallback()** function is used.

Now that we are ready with the video-player's outline, the video file should be loaded. This is achieved using the **cvCaptureFromFile()** function. The next task is to access various properties of this video and then display them at appropriate locations on the *Control* Pannel. To access the video properites **cvGetCapture-Property()** function is used.

If proper codecs are installed and the video consists of atleast one frame, then **cvQueryFrame()** should return the initial frame in the video. If no frame is returned then there must be some problem either with the codecs or the video itself. In such a case, the program is halted with an appropriate error message. If everything goes fine, then the currently grabbed frame is stored into *old_frame*.

Now we come to the task where a frame is grabbed and displayed on the screen. If the player is in *play mode* (i.e. *player* is set to true) then frames are grabbed sequentially at an interval derived from the *FPS* value. The grabbed frame is then resized to the screen_area and displayed to the viewes.

Finally, cleaning up is done by destroying all the open windows and releasing all the images and sub-images.

Parameters

argv[1]	: Video file path

Return values

0	Exit without any problem.
1	Early exit with due to some error.

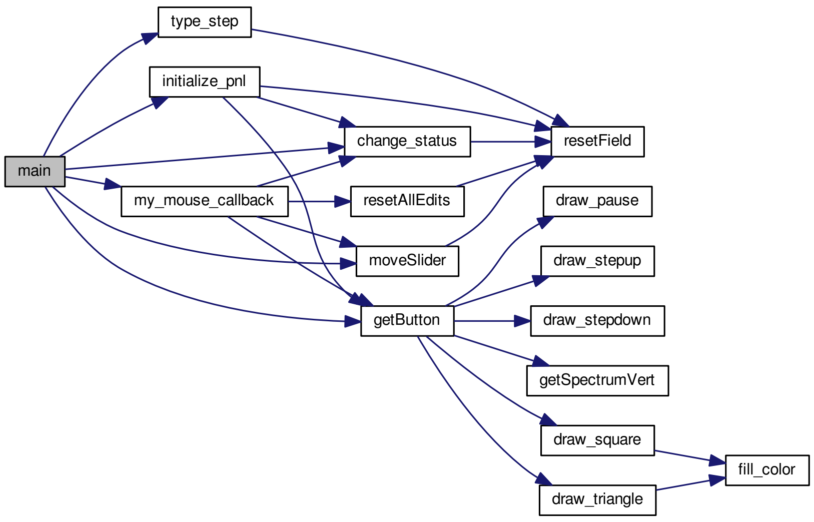
Definition at line 505 of file video_player.c.

```
                                {

  //Initialize the font
  cvInitFont( &font, font_face, hscale, vscale, shear,
thickness, line_type );
  cvInitFont( &font_italic, font_face_italic,
hscale, vscale, shear, thickness, line_type );
  cvInitFont( &font_bold, font_face, hscale, vscale,
shear, thickness+1, line_type );
  cvInitFont( &font_bold_italic, font_face_italic,
hscale, vscale, shear, thickness+1, line_type );
```

```

  //Create the player image
  player = cvCreateImage( cvSize( p_width, p_height ), IPL_DEPTH_8U, 3 );

  //Create Control Pannel
  pnl = cvCreateImageHeader( cvSize( p_width, ctrl_pnl_height ), IPL_DEPTH_8U, 3 );
  pnl->origin = player->origin;
  pnl->widthStep = player->widthStep;
  pnl->imageData = player->imageData + ( p_height - ctrl_pnl_height )*player->widthStep;
  for( int row=0; row<pnl->height; row++ ){
      uchar* ptr = ( uchar* )( pnl->imageData + row*pnl->widthStep );
      for( int col=0; col<pnl->width; col++ ){
          ptr[ col*pnl->nChannels + 0 ] = 226;
          ptr[ col*pnl->nChannels + 1 ] = 235;
          ptr[ col*pnl->nChannels + 2 ] = 240;
      }
  }
  //Add text & buttons
  initialize_pnl( argv[1] );

  //create custom slider (non-opencv)
  slider = cvCreateImageHeader( cvSize( p_width, 10 ), IPL_DEPTH_8U, 3 );
  slider->origin = player->origin;
  slider->widthStep = player->widthStep;
  slider->imageData = player->imageData + ( p_height - sldr_height - ctrl_pnl_height )*player->widthStep;
  for( int row=0; row<slider->height; row++ ){
      uchar* ptr = ( uchar* )( slider->imageData + row*slider->widthStep );
      for( int col=0; col<slider->width; col++ ){
          ptr[ col*slider->nChannels + 0 ] = 94;
          ptr[ col*slider->nChannels + 1 ] = 118;
          ptr[ col*slider->nChannels + 2 ] = 254;
      }
  }
  oslider = cvCloneImage( slider );
  sldr_btn = cvCreateImage( cvSize( 15, sldr_height ), IPL_DEPTH_8U, 3 );
  for( int row=0; row<sldr_btn->height; row++ ){
      uchar* ptr = ( uchar* )( sldr_btn->imageData + row*sldr_btn->widthStep );
      for( int col=0; col<sldr_btn->width; col++ ){
          ptr[ col*slider->nChannels + 0 ] = 100;
          ptr[ col*slider->nChannels + 1 ] = 150;
          ptr[ col*slider->nChannels + 2 ] = 100;
      }
  }
  sldr_val = cvCreateImageHeader( cvSize( sldr_btn_width, sldr_height ), IPL_DEPTH_8U, 3 );
  sldr_val->origin = slider->origin;
  sldr_val->widthStep = slider->widthStep;
  sldr_val->imageData = slider->imageData;
  cvCopy( sldr_btn, sldr_val );

  //display window
  cvNamedWindow( "Video Player", CV_WINDOW_AUTOSIZE );

  //install mouse callback
  cvSetMouseCallback(
      "Video Player",
      my_mouse_callback,
      ( void* )NULL
  );


  //load the video
  vid = cvCaptureFromFile( argv[1] );
  //check the video
  if( !vid ){
      printf( "Error loading the video file. Either missing file or codec not installed\n" );
      return( 1 );
  }
  frame_area = cvCreateImageHeader( cvSize( p_width, scrn_height ), IPL_DEPTH_8U, 3 );
  frame_area->origin = player->origin;
  frame_area->widthStep = player->widthStep;
  frame_area->imageData = player->imageData;
  fps = cvGetCaptureProperty( vid, CV_CAP_PROP_FPS );
  sldr_start = cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES );
  fourcc_l = cvGetCaptureProperty( vid, CV_CAP_PROP_FOURCC );
  fourcc = ( char* )( &fourcc_l );
  sprintf( four_cc_str, "%c%c%c%c", fourcc[0], fourcc[1], fourcc[2], fourcc[3] );
  //printf( "FPS : %f\n", fps );
  sldr_maxval = cvGetCaptureProperty( vid, CV_CAP_PROP_FRAME_COUNT ); //check this property
  if( sldr_maxval<1 ){
      printf( "Number of frames < 1. Cannot continue...\n" );
      return( 1 );
```

```
    }
    cvSetCaptureProperty(
        vid,
        CV_CAP_PROP_POS_FRAMES,
        sldr_start
    );
    sprintf( line, "%d", sldr_maxval );
    cvPutText( numFrames, line, cvPoint( 3, numFrames->height - 4 ), &
  font, black );
    sprintf( line, "%d", ( int )cvRound( fps ) );
    cvPutText( fps_edit, line, cvPoint( 3, fps_edit->height - 4 ), &
  font, black );
    sprintf( line, "%d", sldr_start );
    cvPutText( cur_frame_no, line, cvPoint( 3, cur_frame_no->height - 4 ), &
  font, black );
    sprintf( line, "%s", four_cc_str );
    cvPutText( four_cc_edit, line, cvPoint( 3, four_cc_edit->height - 8 ), &
  font, black );
    moveSlider( sldr_start, OTHER_CALLS );

    frame = cvQueryFrame( vid );
    old_frame = cvCloneImage( frame );
    if( !frame ){
        printf( "Cannot load video. Missing Codec : %s\n", four_cc_str );
        return( 1 );
    }
    cvShowImage( "Video Player", player );

    char c;
    int cur_frame;
    while( 1 ){
        if( ( c = cvWaitKey( 1000/fps ) )==27 ){
            break;
        }
        if( !processing ){
            if( playing ){
                for( int i = 0; i < ( step_val - 1 ); i++ ){
                    cvQueryFrame( vid );
                }
                frame = cvQueryFrame( vid );
                if( !frame ){
                    playing = false;
                }
                else{
                    cvCopy( frame, old_frame );
                }
            }
            //to avoid any negative value of cur_frame
            while( 1 ){
                if( ( cur_frame = ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES ) )>=0 ){
                    break;
                }
                //for some unknown reason cvQueryFrame was needed to be called twice to get to the desired
   frame.
                frame = cvQueryFrame( vid );
                cvCopy( frame, old_frame );
            }
            //defines the task to be carried out when editing a text-field
            if( typing_step ){
                type_step( c, cur_frame );
            }
            //this takes care if for some reason the cur_frame overshoots the sldr_maxval.
            if( cur_frame == ( sldr_maxval-1 ) ){
                getButton( play_pause_btn, PLAY_BTN,
  BTN_ACTIVE );
                sprintf( status_line, "End reached" );
                change_status();
            }
            cvResize( old_frame, frame_area );
            //printf( "Current frame : %d\n", cur_frame );
            moveSlider( cur_frame, OTHER_CALLS );
        }
        cvShowImage( "Video Player", player );
    }

    //destory window
    cvDestroyWindow( "Video Player" );

    //Release image
    cvReleaseImageHeader( &stepdown_btn );
    cvReleaseImageHeader( &stepup_btn );
    cvReleaseImageHeader( &stop_btn );
    cvReleaseImageHeader( &play_pause_btn );
    cvReleaseImageHeader( &step_edit );
    cvReleaseImageHeader( &four_cc_edit );
    cvReleaseImageHeader( &fps_edit );
    cvReleaseImageHeader( &numFrames );
    cvReleaseImageHeader( &cur_frame_no );
    cvReleaseImageHeader( &pnl );
    cvReleaseImageHeader( &sldr_val );
    cvReleaseImageHeader( &slider );
    cvReleaseImageHeader( &frame_area );
    cvReleaseImage( &old_frame );
    cvReleaseImage( &sldr_btn );
    cvReleaseImage( &oslider );
    cvReleaseImage( &player );

    //Release the video
    cvReleaseCapture( &vid );

    return( 0 );
}
```

Here is the call graph for this function:



5.1.3.13 int moveSlider (int *pos*, int *call_from*)

Custome slider's callback function.

Whenever a right-click on our custom-built slider occurs and the mouse is moved over the slider or there is a change in the displayed frame, this function is called. If the function is called from a mouse event then *call_from* is set to MOUSE_CALLBACK and corresponding *pos* indicates the x-coordinate (Cartesian System) of the latest mouse event. The current frame value (*frame_val*) is derived from *pos* using appropriate scaling.

If this function is called from any other function then *call_from* is set to OTHER_CALLS and corresponding *pos* indicates the current frame value which is directly assigned to *frame_val*.

Again scaling is done so that the slider button can be set to an appropriate location between 0 and (p_width - sldr_btn_width)

Proper care is taken so that *frame_val* remains an integeral multiple of *step_val* between 0 and *sldr_maxval*.

Current frame number is then updated in the control pannel and lastly the slider button is set at its appropriate location on the custom-built slider.

Parameters

pos	Either the x-coordinate of the latest mouse event on the slider or the current frame number.
call_from	Set to MOUSE_CALLBACK when this function is called from a mouse callback event, else set to OTHER_CALLS.

Returns

frame_val: The current frame number.

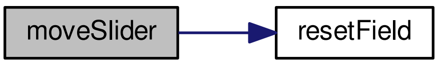
See Also

resetField(), **cvPutText()**, **cvCopy()**

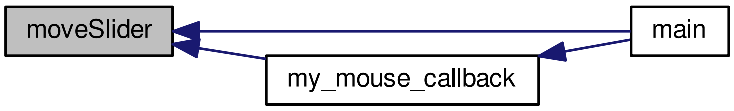
Definition at line 729 of file video_player.c.

```
                                              {
    int frame_val;
    //Scaling to obtain the current frame number
    float scale = ( sldr_maxval )/( float )( p_width );
    //printf( "Pos : %d\tScale : %f\n", pos, scale );
    if( call_from == MOUSE_CALLBACK ){
        frame_val = cvCeil( scale*pos );
    }
    if( call_from == OTHER_CALLS ){
        frame_val = pos;
    }
    //Scaling to set the slider button at an appropriate location between 0 and (p_width - sldr_btn_width)
    scale = ( p_width - sldr_btn_width )/( float )(
  sldr_maxval );
    //printf( "Frame slider : %d\n", frame_val );
    int new_pos = cvCeil( scale*frame_val );
    //frame_val should be an integral multiple of step_val
    if( frame_val%step_val != 0 ){
        frame_val = step_val*( ( int )frame_val/( int )step_val );
    }
    resetField( cur_frame_no, STATIC_TEXT );
    sprintf( line, "%d", frame_val );
    cvPutText( cur_frame_no, line, cvPoint( 3, cur_frame_no->height - 4 ), &font,
  black );
    cvCopy( oslider, slider );
    sldr_val->imageData = slider->imageData + new_pos*slider->nChannels;
    cvCopy( sldr_btn, sldr_val );
    return( frame_val );
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.14 void my_mouse_callback (int *event,* int *x,* int *y,* int *flags,* void ∗ *param*)

Mouse's callback function.

The function is callback for mouse events. Actions to be taken for various mouse events are defined in this function. This function is the 2nd argument to the **`cvSetMouseCallback()`** function.

We associate this mouse callback function for events on the custom-built slider as well as on the different fields in the control pannel. Following is the explaination of the various mouse events used in this scenario and their respective actions. Case1, event = CV_EVENT_MOUSEMOVE i.e. mouse is moved. If the slider button is dragged to a different location, only then this mouse event is to be used to update the frame being displayed. So both conditions viz. the slider is moving (sldr_moving) an the mouse coordinates belong to the custom-built slider are checked and accordingly the new frame number is calculated which is also updated in various fields of the player.

Case2, event = CV_EVENT_LBUTTONDOWN i.e. mouse's left button is pressed down. This event indicates some button being pressed (play, pause, etc or slider button). The mouse coordinates help to identify the button being pressed. Approproate actions on pressing respective buttons are taken.

Case3, event = CV_EVENT_LBUTTONUP i.e. mouse's left button is released after earlier press. Only the slider-button depends on this event it can be dragged along the slider-strip. Therefore, on this event the slider movement is stopped.

See Also

`cvSetMouseCallback()` function for

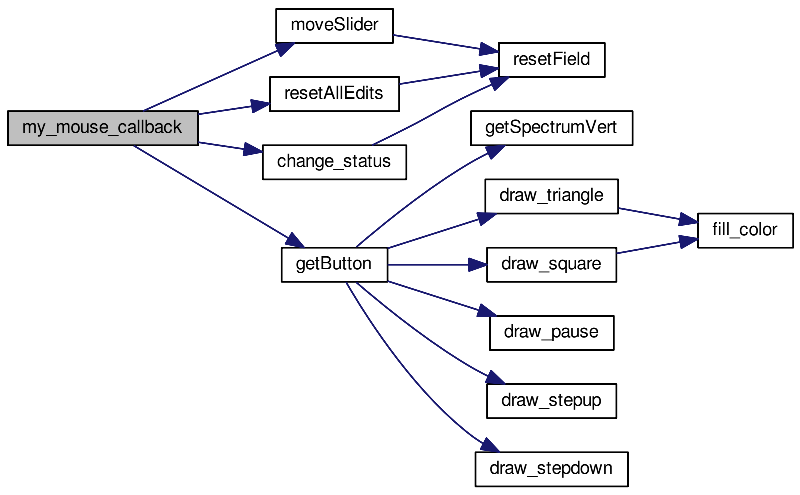
Parameters

event	
x	
y	
flags	
param	

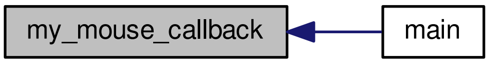
Definition at line 784 of file video_player.c.

```
                                                                           {
    IplImage* image = ( IplImage* )param;
    switch( event ){
        case CV_EVENT_MOUSEMOVE: {
            if( sldr_moving ){
                // mouse on slider
                if( ( y > scrn_height ) && ( y <= scrn_height +
  sldr_height ) ){
                    int cur_frame = moveSlider( x, MOUSE_CALLBACK );
                    if( vid ){
                        cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )( cur_frame-1 ) );
                        cvQueryFrame( vid );
                        cvCopy( cvQueryFrame( vid ), old_frame );
                    }
                }
            }
        }
        break;
        case CV_EVENT_LBUTTONDOWN: {
            sldr_moving = true;
            resetAllEdits();
            // mouse on slider
            if( ( y > scrn_height ) && ( y <= scrn_height +
    sldr_height ) ){
                int cur_frame = moveSlider( x, MOUSE_CALLBACK );
                if( vid ){
                    cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )( cur_frame-1 +
    step_val -1 ) );
                    cvQueryFrame( vid );
                    cvCopy( cvQueryFrame( vid ), old_frame );
                    //printf( "Before val : %f\n", cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES ) );
                }
                if( !playing ){
                    sprintf( status_line, "Slider moved" );
                    change_status();
                }
            }
            // mouse on play/pause button
            if(
                ( y > play_pause_btn_area.y1 ) &&
                ( y <= play_pause_btn_area.y2 ) &&
                ( x > play_pause_btn_area.x1 ) &&
                ( x <= play_pause_btn_area.x2 )
            ){
                //printf( "Frame val : %d\n", ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES ) );
                if( playing ){
                    playing = false;
                    getButton( play_pause_btn, PLAY_BTN,
    BTN_ACTIVE );
                    sprintf( status_line, "Paused" );
                    change_status();
                }
                else{
                    playing = true;
                    getButton( play_pause_btn, PAUSE_BTN,
    BTN_ACTIVE );
                    sprintf( status_line, "Playing" );
                    change_status();
                }
            }
            // mouse on stop button
            if(
                ( y > stop_btn_area.y1 ) &&
                ( y <= stop_btn_area.y2 ) &&
                ( x > stop_btn_area.x1 ) &&
                ( x <= stop_btn_area.x2 )
            ){
                playing = false;
                moveSlider( sldr_start, OTHER_CALLS );
                if( vid ){
                    cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )(
    sldr_start-1 ) );
                    cvQueryFrame( vid );
                    cvCopy( cvQueryFrame( vid ), old_frame );
                }
                getButton( play_pause_btn, PLAY_BTN,
    BTN_ACTIVE );
                sprintf( status_line, "Stopped" );
                change_status();
            }
            // mouse on stepup button
            if(
                ( y > stepup_btn_area.y1 ) &&
                ( y <= stepup_btn_area.y2 ) &&
                ( x > stepup_btn_area.x1 ) &&
                ( x <= stepup_btn_area.x2 )
            ){
                int cur_frame = ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES );
                //printf( "Frame val : %d\n", cur_frame );
                if( cur_frame + 1 + step_val - 1 < sldr_maxval ){
                    for( int i=0; i < ( step_val - 1 ); i++ ){
                        cvQueryFrame( vid );
                    }
                    frame = cvQueryFrame( vid );
                    if( frame ){
                        cvCopy( frame, old_frame );
                    }
                }
                if( !playing ){
                    sprintf( status_line, "Stepped Up" );
                    change_status();
                }
                //printf( "Stepup pressed \n" );
            }
            // mouse on stepdown button
            if(
                ( y > stepdown_btn_area.y1 ) &&
                ( y <= stepdown_btn_area.y2 ) &&
                ( x > stepdown_btn_area.x1 ) &&
                ( x <= stepdown_btn_area.x2 )
            ){
                processing = true;
                int cur_frame = ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES );
                //printf( "Frame val : %d\n", cur_frame );
                if( cur_frame - 1 - ( step_val - 1 ) >= sldr_start ){
                    moveSlider( ( cur_frame - 1 - ( step_val - 1 ) ),
  OTHER_CALLS );
                    cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )( cur_frame - 1 - (
  step_val - 1 ) ) );
                    cvQueryFrame( vid );
                    cvCopy( cvQueryFrame( vid ), old_frame );
                    //printf( "New Frame val : %d\n", ( int )cvGetCaptureProperty( vid,
   CV_CAP_PROP_POS_FRAMES ) );
                }
                if( !playing ){
                    sprintf( status_line, "Stepped Down" );
                    change_status();
                }
                processing = false;
                //printf( "Stepdown pressed \n" );
            }
            // mouse on step_edit field
            if(
                ( y > step_edit_area.y1 ) &&
                ( y <= step_edit_area.y2 ) &&
                ( x > step_edit_area.x1 ) &&
                ( x <= step_edit_area.x2 )
            ){
                sprintf( edit_text, "" );
                typing_step = true;
            }
        }
        break;
        case CV_EVENT_LBUTTONUP: {
            sldr_moving = false;
        }
        break;
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



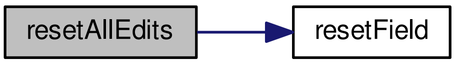
5.1.3.15 void resetAllEdits ( )

Function to reset all fields to their previous contents.

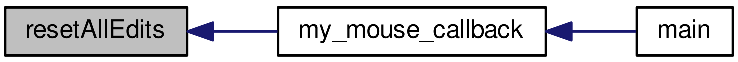
Definition at line 1466 of file video_player.c.

```
                      {
    resetField( step_edit, EDIT_TEXT );
    sprintf( edit_text, "%d", step_val );
    cvPutText( step_edit, edit_text, cvPoint( 3, step_edit->height - 4 ), &
 font, black );
    typing_step = false;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.16 void resetField (IplImage ∗ *image,* int *text_type*)

Function to reset a given text field.

This function will reset a text-field (a textbox or a static text). The text-field is nothing but a sub-image. Therefore all the pixel values are to be reset to the original values of the respecitve text fields (white with black border for EDIT_TEXT and Control Pannel's color with black border for STATIC_TEXT).

Whenever the value in the text-field is changed, the text-field being an image, the new value is overwritten over the old value. Therefore, every time a new value is to be written, the respective field need to be reset.

Parameters

image	: The sub-image (i.e. the text-field) to be reset.
text_type	: Either STATIC_TEXT or EDIT_TEXT.

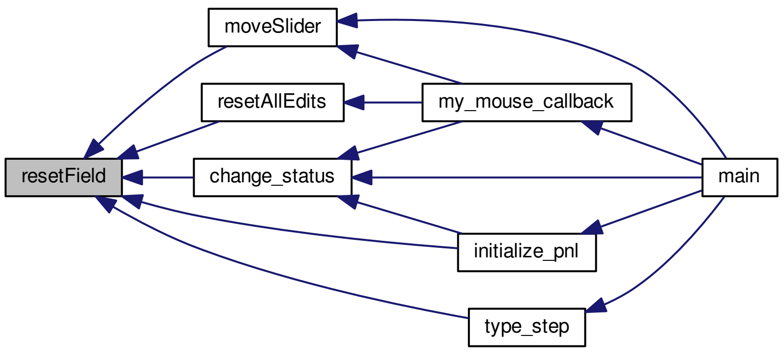
See Also

IplImage

Definition at line 952 of file video_player.c.

```
                                              {
    if( text_type == STATIC_TEXT ){
        for( int row=0; row<image->height; row++ ){
            uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
            for( int col=0; col<image->width; col++  ){
                ptr[ col*image->nChannels + 0 ] = 226;
                ptr[ col*image->nChannels + 1 ] = 235;
                ptr[ col*image->nChannels + 2 ] = 240;
            }
        }
    }
    else{
        for( int row=0; row<image->height; row++ ){
            uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
            for( int col=0; col<image->width; col++  ){
                if( row==0 || row==image->height-1 || col==0 || col==image->width-1 ){
                    ptr[ col*image->nChannels + 0 ] = 0;
                    ptr[ col*image->nChannels + 1 ] = 0;
                    ptr[ col*image->nChannels + 2 ] = 0;
                }
                else
                {
                    ptr[ col*image->nChannels + 0 ] = 255;
                    ptr[ col*image->nChannels + 1 ] = 255;
                    ptr[ col*image->nChannels + 2 ] = 255;
                }
            }
        }
    }
}
```

Here is the caller graph for this function:



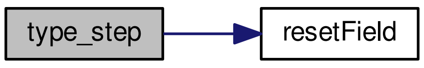
5.1.3.17 void type_step (char *c*, int *frame_val*)

Function to edit a textbox.

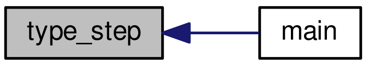
Definition at line 1411 of file video_player.c.

```
                                       {
    resetField( step_edit, EDIT_TEXT );
    char temp_text[ 20 ];
    int cur_frame;
    sprintf( temp_text, "" );
    if( blinking ){
        if( blink_count<blink_max ){
            blink_count++;
        }
        else{
            blinking = false;
            blink_char = ' ';
            blink_count = 0;
        }
        //printf( "Blinking...\n" );
    }
    else{
        if( blink_count<blink_max ){
            blink_count++;
        }
        else{
            blinking = true;
            blink_char = '|';
            blink_count = 0;
        }
        //printf( "Not blinking...\n" );
    }
    //valid number
    if( c>=48 && c<=57 ){
        sprintf( temp_text, "%s%c", edit_text, c );
        if( ( frame_val + atoi( temp_text ) )>=0 && ( frame_val + atoi( temp_text ) )<= sldr_maxval && ( atoi( temp_text )!=0 ) ){
            sprintf( edit_text, "%s", temp_text );
        }
    }
    //backspace
    if( c==8 ){
        if( strcmp( edit_text, "" )!=0 ){
            for( int count=0; count<( strlen( edit_text )-1 ); count++ ){
                sprintf( temp_text, "%s%c", temp_text, edit_text[ count ] );
            }
            sprintf( edit_text, "%s", temp_text );
        }
    }
    sprintf( temp_text, "%s%c", edit_text, blink_char );
    cvPutText( step_edit, temp_text, cvPoint( 3, step_edit->height - 4 ), & font, black );
    if( c==10 ){
        resetField( step_edit, EDIT_TEXT );
        cvPutText( step_edit, edit_text, cvPoint( 3, step_edit->height - 4 ), & font, black );
        step_val = atoi( edit_text );
        //printf( "Step : %d\n", step );
        typing_step = false;
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4 Variable Documentation

5.1.4.1 CvScalar black = cvScalar(0, 0, 0)

Black color.

Definition at line 426 of file video_player.c.

5.1.4.2 char blink_char = '|'

Threshold to toogle the *blink_char*.

Definition at line 403 of file video_player.c.

5.1.4.3 int blink_count = 0

Blinker count.

This counter is used to toogle the blinker character blink_char. Whenever this counter crosses blink_max, the *blink_char* is toogled.

See Also

type_step().

Definition at line 400 of file video_player.c.

5.1.4.4 int blink_max = 5

Definition at line 402 of file video_player.c.

5.1.4.5 bool blinking = false

True when blinking character is set.

Definition at line 419 of file video_player.c.

5.1.4.6 CvScalar blue = cvScalar(255, 0, 0)

Blue color.

Definition at line 425 of file video_player.c.

5.1.4.7 CvScalar brown = cvScalar(0, 0, 127)

Brown color.

Definition at line 433 of file video_player.c.

5.1.4.8 IplImage∗ cur_frame_no

Pointer to current frame number static-text.

Points to the sub-image showing the current frame number.

See Also

`IplImage`, initialize_pnl(), moveSlider().

Definition at line 281 of file video_player.c.

5.1.4.9 char edit_text[20]

Memory to hold a textbox string temporarily.

This will hold a textbox string temporarily. Whenever a textbox is to be used, the original string in the textbox is required while editing its contents. This is the primary use of this memory.

Definition at line 372 of file video_player.c.

5.1.4.10 CvFont font

Normal font.

Definition at line 436 of file video_player.c.

5.1.4.11 CvFont font_bold

Bold font.

Definition at line 438 of file video_player.c.

5.1.4.12 CvFont font_bold_italic

Bold Italic font.

Definition at line 439 of file video_player.c.

5.1.4.13 int font_face = CV_FONT_HERSHEY_SIMPLEX

Font face.

Definition at line 441 of file video_player.c.

5.1.4.14 int font_face_italic = CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC

Font face.

Definition at line 440 of file video_player.c.

5.1.4.15 CvFont font_italic

Italic font.

Definition at line 437 of file video_player.c.

5.1.4.16 IplImage∗ four_cc_edit

Pointer to FOUR_CC static-text.

Points to the sub-image showing FOUR_CC static text.

See Also

`IplImage`, initialize_pnl().

Definition at line 297 of file video_player.c.

5.1.4.17 Field_Area four_cc_edit_area

FOUR_CC static-text coordinates.

Definition at line 410 of file video_player.c.

5.1.4.18 char four_cc_str[4]

Memory to hold the Four Character Code (FOUR_CC).

Definition at line 375 of file video_player.c.

5.1.4.19 char∗ fourcc

Four_CC temporary string.

An intermediate string to hold the FOUR_CC value while parsing from fourcc_l to four_cc_str.

Definition at line 392 of file video_player.c.

5.1.4.20 long fourcc_l

Four Character Code.

Hold the FOUR_CC value in double format. This value is directly read from the input video file, parsed to a string (using fourcc) and stored to four_cc_str.

Definition at line 387 of file video_player.c.

5.1.4.21 double fps

Frames per second.

Frames Per Second value is stored in this variable. This value is read from the input video file.

Definition at line 381 of file video_player.c.

5.1.4.22 IplImage∗ fps_edit

Pointer to FPS (Frames Per Second) static-text.

Points to the sub-image showing the FPS. This is currently a static-text field and its value is to be loaded from the video initially. Later, the functionality to edit this field can be added, therefore the pointer has "edit" in its name. It hold the value of fps.

See Also

`IplImage`, initialize_pnl().

Definition at line 289 of file video_player.c.

5.1.4.23 Field_Area fps_edit_area

FPS static-text coordinates.

Definition at line 409 of file video_player.c.

5.1.4.24 IplImage∗ frame

Pointer to the fetched frame sub-image.

This will point to frame fetched using `cvQueryFrame()`. Therefore, this pointer is only declared and not defined. The allocation and deallocation of memory pointed by this pointer is handled by `cvQueryFrame()`.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 266 of file video_player.c.

5.1.4.25 IplImage∗ frame_area

Pointer to the frame-area sub-image.

The frame-area sub-image is originally created as an empty image using the `cvCreateImage()` function. Here the currently fetched frame will be displayed. It is then assigned the following properties of the main image so that it becomes a sub-image (or region of interest).

- *origin* = `origin` of main image.
- *widthStep* = `widthStep` of main image.
- *data* origin location = `desired data location` from the main image

Once this assignment is done, any change in this Frame-area sub-image, will be reflected directly on the screen.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 258 of file video_player.c.

5.1.4.26 CvScalar gray = cvScalar(242, 242, 242)

Gray color.

Definition at line 430 of file video_player.c.

5.1.4.27 CvScalar green = cvScalar(0, 255, 0)

Green color.

Definition at line 424 of file video_player.c.

5.1.4.28 double hscale = 0.5

Font's Horizontal Scale parameter.

Definition at line 442 of file video_player.c.

5.1.4.29 CvScalar light_yellow = cvScalar(242, 255, 255)

Light Yellow color.

Definition at line 428 of file video_player.c.

5.1.4.30 char line[20]

Memory to hold any string temporarily.

Definition at line 366 of file video_player.c.

5.1.4.31 int line_type = 8

Font's Line-type parameter.

Definition at line 446 of file video_player.c.

5.1.4.32 IplImage∗ numFrames

Pointer to Total Frames static-text.

Points to the sub-image showing the Total Number of Frames static-text. It holds the value of sldr_maxval.

See Also

`IplImage`, initialize_pnl().

Definition at line 313 of file video_player.c.

5.1.4.33 IplImage∗ old_frame

Pointer to the previously fetched frame.

The current fetched frame using `cvQueryFrame()` is cloned to *old_frame* before fetching the next frame. Thus, this pointer points to an `IplImage` structure holding the previously fetched frame.

See Also

`cvLoadImage()`, `cvReleaseImage()`.

Definition at line 274 of file video_player.c.

5.1.4.34 CvScalar orange = cvScalar(0, 242, 255)

Orange color.

Definition at line 431 of file video_player.c.

5.1.4.35 IplImage∗ oslider

Pointer to temporary slider-value static-text sub-image.

The temporary slider-value static-text sub-image is originally created as an empty image using the `cvCreateImage()` function. This is used to temporarily store the original slider-value. It is then assigned the following properties of the main image so that it becomes a sub-image (or region of interest).

- *origin* = `origin` of main image.
- *widthStep* = `widthStep` of main image.
- *data* origin location = `desired data location` from the main image

Once this assignment is done, any change in this temporary slider-value static-text sub-image, will be reflected directly on the screen.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 244 of file video_player.c.

5.1.4.36 IplImage∗ play_pause_btn

Pointer to play/pause button area.

Points to the sub-image having the play / pause button.

See Also

`IplImage`, initialize_pnl().

Definition at line 330 of file video_player.c.

5.1.4.37 Field_Area play_pause_btn_area

The blinking character, toogled with an underscore (_).

Play / Pause Button coordinates.

Definition at line 405 of file video_player.c.

5.1.4.38 IplImage∗ player

Pointer to the main image.

Pointer to the main image shown on the screen. The various buttons, screen-area etc are sub-images of this image. Initially this image is created as an empty image using the `cvCreateImage()` function. Later, every sub-image's data part is assigned the desired part of this main image. Now, any further operation on the sub-images reflects the change in this image as well.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 174 of file video_player.c.

5.1.4.39 bool playing = false

True when the video is being played.

Definition at line 416 of file video_player.c.

5.1.4.40 IplImage∗ pnl

Pointer to the control-pannel sub-image.

The control-pannel sub-image is originally created as an empty image using the `cvCreateImage()` function. It is then assigned the following properties of the main image so that it becomes a sub-image (or region of interest).

- *origin* = `origin` of main image.
- *widthStep* = `widthStep` of main image.
- *data* origin location = `desired data location` from the main image

Once this assignment is done, any change in this Control-Pannel sub-image, will be reflected directly on the screen.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 188 of file video_player.c.

5.1.4.41 bool processing = false

True when some processing is carried out.

Definition at line 417 of file video_player.c.

5.1.4.42 CvScalar red = cvScalar(0, 0, 255)

Red color.

Definition at line 423 of file video_player.c.

5.1.4.43 double shear = 0

Font's Shear parameter.

Definition at line 444 of file video_player.c.

5.1.4.44 IplImage∗ sldr_btn

Pointer to the slider-button sub-image.

The slider-button sub-image is originally created as an empty image using the `cvCreateImage()` function. It is then assigned the following properties of the main image so that it becomes a sub-image (or region of interest).

- *origin* = `origin` of main image.
- *widthStep* = `widthStep` of main image.
- *data* origin location = `desired data location` from the main image

Once this assignment is done, any change in this Slider-Button sub-image, will be reflected directly on the screen.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 216 of file video_player.c.

5.1.4.45 int sldr_maxval

The maximum number of frames in the video.

Definition at line 358 of file video_player.c.

5.1.4.46 bool sldr_moving = false

Ture when slider is moving.

Definition at line 415 of file video_player.c.

5.1.4.47 int sldr_start

Indicates the starting position (frame number) of the slider.

Definition at line 357 of file video_player.c.

5.1.4.48 IplImage∗ sldr_val

Pointer to the slider-value static-text sub-image.

The slider-value static-text sub-image is originally created as an empty image using the `cvCreateImage()` function. It is then assigned the following properties of the main image so that it becomes a sub-image (or region of interest).

- *origin* = `origin` of main image.
- *widthStep* = `widthStep` of main image.
- *data* origin location = `desired data location` from the main image

Once this assignment is done, any change in this Slider-value Static-Text sub-image, will be reflected directly on the screen.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 230 of file video_player.c.

5.1.4.49 IplImage∗ slider

Pointer to the slider-strip sub-image.

The slider-strip sub-image is originally created as an empty image using the `cvCreateImage()` function. It is then assigned the following properties of the main image so that it becomes a sub-image (or region of interest).

- *origin* = `origin` of main image.
- *widthStep* = `widthStep` of main image.
- *data* origin location = `desired data location` from the main image

Once this assignment is done, any change in this Slider-strip sub-image, will be reflected directly on the screen.

See Also

`IplImage`, `cvLoadImage()`, `cvReleaseImage()`.

Definition at line 202 of file video_player.c.

5.1.4.50 IplImage* status_edit

Pointer to "Status" static-text.

Points to the sub-image showing the status static-text. Holds the string in status_line.

See Also

`IplImage`, initialize_pnl().

Definition at line 305 of file video_player.c.

5.1.4.51 Field_Area status_edit_area

Status string coordinates.

Definition at line 411 of file video_player.c.

5.1.4.52 char status_line[15]

Memory to hold the "status" string.

Definition at line 374 of file video_player.c.

5.1.4.53 IplImage* step_edit

Pointer to the Step textbox.

Points to the sub-image showing the Step textbox. This will hold the value of step_val.

See Also

`IplImage`, initialize_pnl().

Definition at line 321 of file video_player.c.

5.1.4.54 Field_Area step_edit_area

Step textbox coordinates.

Definition at line 412 of file video_player.c.

5.1.4.55 int step_val = 1

Step size.

The step size is the distance between the current and the next frame to be fetched. To view the video as it is, every frame has to be displayed. Therefore, by default this value is set to 1.

Definition at line 364 of file video_player.c.

5.1.4.56 IplImage* stepdown_btn

Pointer to step_down button area.

Points to the sub-image having the step_down button.

See Also

`IplImage`, initialize_pnl().

Definition at line 354 of file video_player.c.

5.1.4.57 Field_Area stepdown_btn_area

Step Down Button coordinates.

Definition at line 408 of file video_player.c.

5.1.4.58 IplImage* stepup_btn

Pointer to step_up button area.

Points to the sub-image having the step_up button.

See Also

`IplImage`, initialize_pnl().

Definition at line 346 of file video_player.c.

5.1.4.59 Field_Area stepup_btn_area

Step Up Button coordinates.

Definition at line 407 of file video_player.c.

5.1.4.60 IplImage* stop_btn

Pointer to stop button area.

Points to the sub-image having the stop button.

See Also

`IplImage`, initialize_pnl().

Definition at line 338 of file video_player.c.

5.1.4.61 Field_Area stop_btn_area

Stop Button coordinates.

Definition at line 406 of file video_player.c.

5.1.4.62 int thickness = 1

Font's Thickness parameter.

Definition at line 445 of file video_player.c.

5.1.4.63 bool typing_step = false

True when any textbox value is being edited.

Definition at line 418 of file video_player.c.

5.1.4.64 CvCapture∗ vid

Pointer to CvCapture structure.

A global pointer to the CvCapture structure is created so that the capture properties can be extracted and edited seamlessly from any of the related functions. CvCapture is basically used to capture the video into the program using the functions `cvCaptureFromFile()` (for capturing from file) or `cvCaptureFromCAM()` (for capturing directly from the attached camera). The details of CvCapture structure can be found `here`.

See Also

`cvReleaseCapture()`.

Definition at line 166 of file video_player.c.

5.1.4.65 CvScalar voilet = cvScalar(255, 0, 127)

Voilet color.

Definition at line 432 of file video_player.c.

5.1.4.66 double vscale = 0.5

Font's Vertical Scale parameter.

Definition at line 443 of file video_player.c.

5.1.4.67 CvScalar white = cvScalar(255, 255, 255)

White color.

Definition at line 427 of file video_player.c.

5.1.4.68 CvScalar yellow = cvScalar(0, 255, 255)

Yellow color.

Definition at line 429 of file video_player.c.

Index