

My Video Player

Generated by Doxygen 1.8.6

Sat Jan 3 2015 01:44:24

Contents

- 1 Simple Video Player Using OpenCV** **1**
 - 1.1 Abstract 1

- 2 Data Structure Index** **3**
 - 2.1 Data Structures 3

- 3 File Index** **5**
 - 3.1 File List 5

- 4 Data Structure Documentation** **7**
 - 4.1 Field_Area Struct Reference 7
 - 4.1.1 Detailed Description 7
 - 4.1.2 Field Documentation 7
 - 4.1.2.1 x1 7
 - 4.1.2.2 x2 7
 - 4.1.2.3 y1 8
 - 4.1.2.4 y2 8

- 5 File Documentation** **9**
 - 5.1 video_player.c File Reference 9
 - 5.1.1 Detailed Description 14
 - 5.1.2 Macro Definition Documentation 14
 - 5.1.2.1 BTN_ACTIVE 14
 - 5.1.2.2 BTN_INACTIVE 14
 - 5.1.2.3 ctrl_pnl_height 14
 - 5.1.2.4 EDIT_CALLS 14
 - 5.1.2.5 EDIT_TEXT 14
 - 5.1.2.6 MOUSE_CALLBACK 14
 - 5.1.2.7 OTHER_CALLS 15
 - 5.1.2.8 p_height 15
 - 5.1.2.9 p_width 15
 - 5.1.2.10 PAUSE_BTN 15
 - 5.1.2.11 PLAY_BTN 15

5.1.2.12	scrn_height	16
5.1.2.13	sldr_btn_width	16
5.1.2.14	sldr_height	16
5.1.2.15	STATIC_TEXT	16
5.1.2.16	STEPDOWN_BTN	16
5.1.2.17	STEPUP_BTN	16
5.1.2.18	STOP_BTN	16
5.1.3	Function Documentation	17
5.1.3.1	change_status	17
5.1.3.2	draw_pause	17
5.1.3.3	draw_square	18
5.1.3.4	draw_stepdown	19
5.1.3.5	draw_stepup	20
5.1.3.6	draw_triangle	20
5.1.3.7	fill_color	21
5.1.3.8	getButton	22
5.1.3.9	getSpectrumHorz	23
5.1.3.10	getSpectrumVert	24
5.1.3.11	initialize_pnl	25
5.1.3.12	main	28
5.1.3.13	moveSlider	32
5.1.3.14	my_mouse_callback	34
5.1.3.15	resetAllEdits	37
5.1.3.16	resetField	37
5.1.3.17	type_step	39
5.1.4	Variable Documentation	41
5.1.4.1	black	41
5.1.4.2	blink_char	41
5.1.4.3	blink_count	41
5.1.4.4	blink_max	41
5.1.4.5	blinking	41
5.1.4.6	blue	41
5.1.4.7	brown	42
5.1.4.8	cur_frame_no	42
5.1.4.9	edit_text	42
5.1.4.10	font	42
5.1.4.11	font_bold	42
5.1.4.12	font_bold_italic	42
5.1.4.13	font_face	42
5.1.4.14	font_face_italic	42

5.1.4.15	font_italic	43
5.1.4.16	four_cc_edit	43
5.1.4.17	four_cc_edit_area	43
5.1.4.18	four_cc_str	43
5.1.4.19	fourcc	43
5.1.4.20	fourcc_l	43
5.1.4.21	fps	43
5.1.4.22	fps_edit	43
5.1.4.23	fps_edit_area	44
5.1.4.24	frame	44
5.1.4.25	frame_area	44
5.1.4.26	gray	44
5.1.4.27	green	45
5.1.4.28	hscale	45
5.1.4.29	light_yellow	45
5.1.4.30	line	45
5.1.4.31	line_type	45
5.1.4.32	numFrames	45
5.1.4.33	old_frame	45
5.1.4.34	orange	46
5.1.4.35	oslider	46
5.1.4.36	play_pause_btn	46
5.1.4.37	play_pause_btn_area	46
5.1.4.38	player	46
5.1.4.39	playing	47
5.1.4.40	pnl	47
5.1.4.41	processing	47
5.1.4.42	red	47
5.1.4.43	shear	47
5.1.4.44	sldr_btn	47
5.1.4.45	sldr_maxval	48
5.1.4.46	sldr_moving	48
5.1.4.47	sldr_start	48
5.1.4.48	sldr_val	48
5.1.4.49	slider	48
5.1.4.50	status_edit	49
5.1.4.51	status_edit_area	49
5.1.4.52	status_line	49
5.1.4.53	step_edit	49
5.1.4.54	step_edit_area	49

5.1.4.55	step_val	49
5.1.4.56	stepdown_btn	50
5.1.4.57	stepdown_btn_area	50
5.1.4.58	stepup_btn	50
5.1.4.59	stepup_btn_area	50
5.1.4.60	stop_btn	50
5.1.4.61	stop_btn_area	50
5.1.4.62	thickness	50
5.1.4.63	typing_step	51
5.1.4.64	vid	51
5.1.4.65	voilet	51
5.1.4.66	vscale	51
5.1.4.67	white	51
5.1.4.68	yellow	51
Index		52

Chapter 1

Simple Video Player Using OpenCV



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:

[Field_Area](#)

Structure to store the top-left and bottom-right corner coordinates of various fields & buttons . . . 7

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

[video_player.c](#) 9

Chapter 4

Data Structure Documentation

4.1 Field_Area Struct Reference

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

Data Fields

- `int x1`
x coordinate of the top-left corner.
- `int y1`
y coordinate of the top-left corner.
- `int x2`
x coordinate of the bottom-right corner.
- `int y2`
y coordinate of the bottom-right corner.

4.1.1 Detailed Description

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

At times, it is necessary to know if the mouse is pointing to a specific area in the displayed image. Since all the buttons, text-fields, slider etc. are nothing but 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 coordinates 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 corner.

Definition at line 153 of file `video_player.c`.

4.1.2.2 `int x2`

x coordinate of the bottom-right corner.

Definition at line 155 of file video_player.c.

4.1.2.3 int y1

y coordinate of the top-left corner.

Definition at line 154 of file video_player.c.

4.1.2.4 int y2

y coordinate of the bottom-right corner.

Definition at line 156 of file video_player.c.

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

- [video_player.c](#)

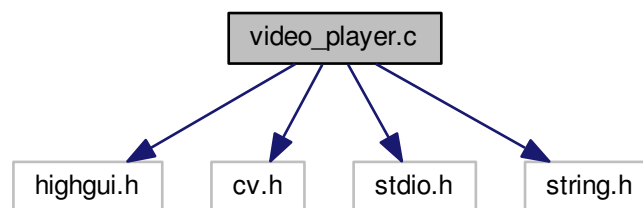
Chapter 5

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 bottom-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 Panel'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 horizontally 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.*
- `lpImage * stop_btn`
 - Pointer to stop button area.*
- `lpImage * stepup_btn`
 - Pointer to step_up button area.*
- `lpImage * 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 toggle the blink_char.*
- `Field_Area play_pause_btn_area`
 - The blinking character, toggled 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). Curretnly, 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). Currently, 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 convenience, 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.

```

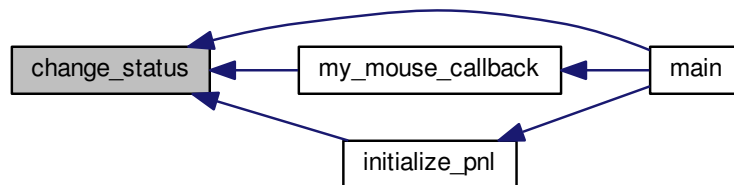
1405         {
1406     resetField( status_edit, STATIC_TEXT );
1407     cvPutText( status_edit, status_line, cvPoint( 3,
1408     status_edit->height - 8 ), &font, black );

```

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

<i>image</i>	: The image where we want to place the pause-rectangles
--------------	---

<i>color</i>	: The desired color
--------------	---------------------

See Also

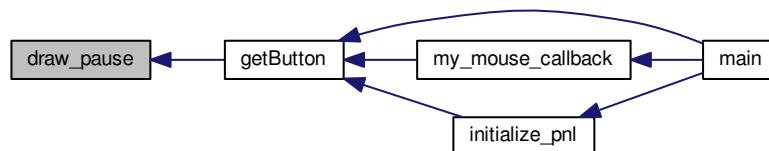
[cvPoint\(\)](#), [cvLine\(\)](#).

Definition at line 1305 of file video_player.c.

```

1305                                     {
1306     int y_start = 3;
1307     int y_end = image->height - y_start;
1308     int dist = 3;
1309     CvPoint pt1, pt2, tmp1, tmp2;
1310     pt1.x = image->width/2;
1311     pt1.y = y_start;
1312     pt2.x = pt1.x;
1313     pt2.y = y_end;
1314     for( int col=0; col<5; col++ ){
1315         tmp1.x = pt1.x + dist + col;
1316         tmp1.y = pt1.y;
1317         tmp2.x = pt2.x + dist + col;
1318         tmp2.y = pt2.y;
1319         cvLine( image, tmp1, tmp2, color );
1320         tmp1.x = pt1.x - dist - col;
1321         tmp1.y = pt1.y;
1322         tmp2.x = pt2.x - dist - col;
1323         tmp2.y = pt2.y;
1324         cvLine( image, tmp1, tmp2, color );
1325     }
1326 }
```

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

<i>image</i>	: The image where we want to place the stop-square
<i>color</i>	: The desired color

See Also

[cvPoint\(\)](#), [cvRectangle](#), [fill_color](#).

Definition at line 1285 of file video_player.c.


```

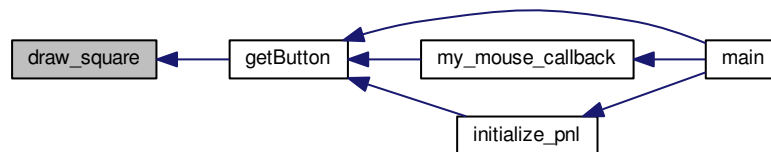
1285         {
1286         CvPoint pt1, pt2;
1287         pt1.x = 3*image->width/8;
1288         pt1.y = 3;
1289         pt2.x = 5*image->width/8;
1290         pt2.y = image->height - pt1.y;
1291         cvRectangle( image, pt1, pt2, color );
1292         fill_color( image, color );
1293     }

```

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.

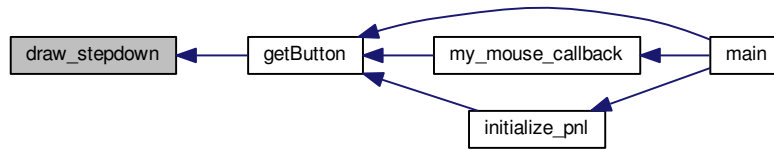
Definition at line 1382 of file video_player.c.

```

1382         {
1383         CvPoint pt1, pt2, pt3, pt4;
1384         pt1.x = 4*image->width/8;
1385         pt2.x = 6*image->width/8;
1386         pt3.x = pt1.x;
1387         pt4.x = pt2.x;
1388         int y_start = 3;
1389         int y_end = image->height/2 ;
1390         for( int row=y_start; row<=y_end; row++ ){
1391             pt1.x = pt1.x - row + y_start;
1392             pt2.x = pt2.x - row + y_start;
1393             pt1.y = row;
1394             pt2.y = row;
1395             pt3.x = pt1.x;
1396             pt4.x = pt2.x;
1397             pt3.y = image->height - row;
1398             pt4.y = pt3.y;
1399             cvLine( image, pt1, pt2, color );
1400             cvLine( image, pt3, pt4, color );
1401         }
1402     }

```

Here is the caller graph for this function:



5.1.3.5 void draw_stepup (IplImage * image, CvScalar color)

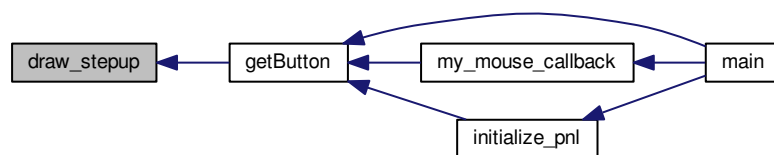
Function to draw a step-up symbol on a given image.

Definition at line 1359 of file video_player.c.

```

1359     {
1360     CvPoint pt1, pt2, pt3, pt4;
1361     pt1.x = 2*image->width/8;
1362     pt2.x = 4*image->width/8;
1363     pt3.x = pt1.x;
1364     pt4.x = pt2.x;
1365     int y_start = 3;
1366     int y_end = image->height/2 ;
1367     for( int row=y_start; row<=y_end; row++ ){
1368         pt1.x = pt1.x + row - y_start;
1369         pt2.x = pt2.x + row - y_start;
1370         pt1.y = row;
1371         pt2.y = row;
1372         pt3.x = pt1.x;
1373         pt4.x = pt2.x;
1374         pt3.y = image->height - row;
1375         pt4.y = pt3.y;
1376         cvLine( image, pt1, pt2, color );
1377         cvLine( image, pt3, pt4, color );
1378     }
1379 }
  
```

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

<i>image</i>	: The image where we want to place the play-triangle
<i>color</i>	: The desired color

See Also

[cvPoint\(\)](#), [cvLine\(\)](#), [fill_color](#).

Definition at line 1261 of file video_player.c.

```

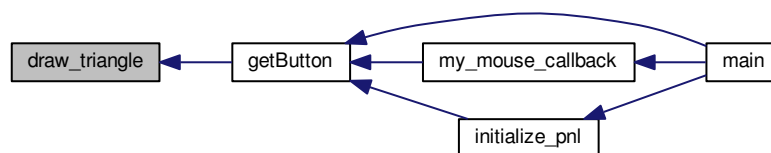
1261                                     {
1262     CvPoint pt1, pt2, pt3;
1263     pt1.x = image->width/3;
1264     pt1.y = 3;
1265     pt2.x = pt1.x;
1266     pt2.y = image->height - pt1.y;
1267     pt3.x = 2*pt1.x;
1268     pt3.y = image->height/2;
1269     cvLine( image, pt1, pt2, color );
1270     cvLine( image, pt3, pt2, color );
1271     cvLine( image, pt1, pt3, color );
1272     fill_color( image, color );
1273 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



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.

```

1331                                     {
1332     bool start_fill = false;
1333     for( int row=0; row<image->height; row++ ){
1334         uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
1335         for( int col=0; col<image->width; col++ ){

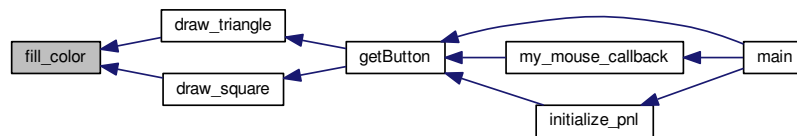
```

```

1336         if(
1337             ( ptr[ col*image->nChannels + 0 ] == color.val[0] ) &&
1338             ( ptr[ col*image->nChannels + 1 ] == color.val[1] ) &&
1339             ( ptr[ col*image->nChannels + 2 ] == color.val[2] )
1340         ){
1341             if( !start_fill ){
1342                 start_fill = true;
1343             }
1344             else{
1345                 start_fill = false;
1346                 break;
1347             }
1348         }
1349         if( start_fill ){
1350             ptr[ col*image->nChannels + 0 ] = color.val[0];
1351             ptr[ col*image->nChannels + 1 ] = color.val[1];
1352             ptr[ col*image->nChannels + 2 ] = color.val[2];
1353         }
1354     }
1355 }
1356 }

```

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

<i>image</i>	: This is the sub-image for the desired button.
<i>btn_type</i>	: Can be any of the following viz. PLAY_BTN, PAUSE_BTN, STOP_BTN, STEPUP_BTN, STEPDOWN_BTN.
<i>btn_state</i>	: 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.

```

991
992     getSpectrumVert( image, violet, black );
993     if( btn_type==PLAY_BTN ){
994         draw_triangle( image, green );
995     }
996     if( btn_type==STOP_BTN ){
997         draw_square( image, green );
998     }
999     if( btn_type==PAUSE_BTN ){
1000         draw_pause( image, green );
1001     }
1002     if( btn_type==STEPUP_BTN ){
1003         draw_stepup( image, green );

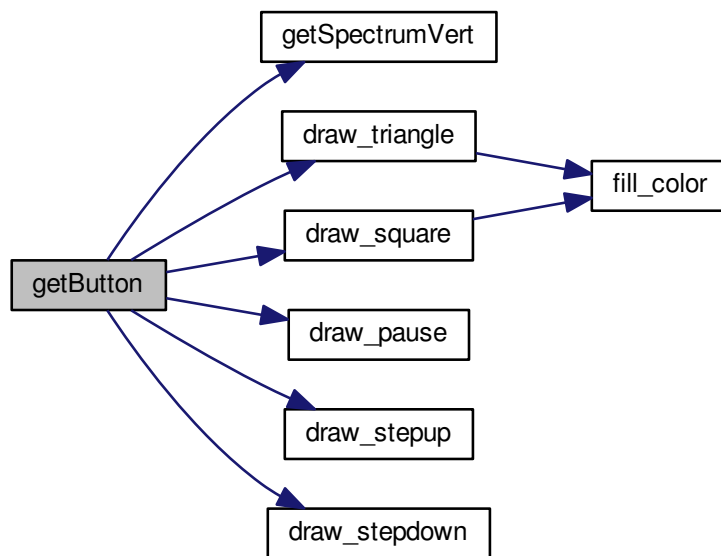
```

```

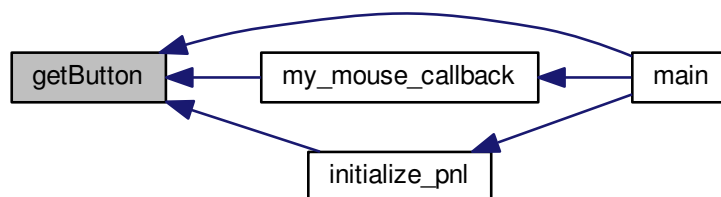
1004     }
1005     if( btn_type==STEPDOWN_BTN ){
1006         draw_stepdown( image, green );
1007     }
1008 }

```

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 horizontally color a button.

This function is nothing by an implementation of linear interpolation 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

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

See Also

IplImage

Definition at line 1084 of file video_player.c.

```

1084                                     {
1085     //Color the leftmost and rightmost pixels of each row with with color1 and color2 respectively
1086     for( int row=0; row<image->height; row++){
1087         uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
1088         for( int chl=0; chl<image->nChannels; chl++){
1089             ptr[chl] = color1.val[chl];
1090             ptr[ ( image->width-1 )*image->nChannels + chl ] = color2.val[ chl ];
1091         }
1092     }
1093     //Interploation applied here
1094     //b_a_L => (B-A)/L... (X-A)/l = (B-A)/L :: => X = ((B-A)/L)*l + A
1095     for( int row=0; row<image->height; row++){
1096         uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
1097         for( int col=0; col<image->width; col++){
1098             for( int chl=0; chl<image->nChannels; chl++){
1099                 ptr[ col*image->nChannels + chl ] = ( ptr[ ( image->width - 1 )*image->nChannels + chl ]
- ptr[ chl ] )*( col/( float )image->width ) + ptr[ chl ];
1100             }
1101         }
1102     }
1103 }

```

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 interpolation 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

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

See Also

[IplImage](#)

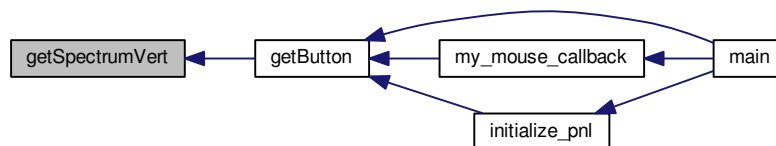
Definition at line 1029 of file video_player.c.

```

1029                                     {
1030     for( int row=0; row<image->height; row++ ){
1031         //If topmost row is selected
1032         if( row==0 ){
1033             uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
1034             for( int col=0; col<image->width; col++ ){
1035                 for( int chl=0; chl<image->nChannels; chl++ ){
1036                     ptr[ col*image->nChannels + chl ] = color1.val[chl];
1037                 }
1038             }
1039         }
1040         //If bottommost row is selected
1041         if( row==image->height-1 ){
1042             uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
1043             for( int col=0; col<image->width; col++ ){
1044                 for( int chl=0; chl<image->nChannels; chl++ ){
1045                     ptr[ col*image->nChannels + chl ] = color2.val[chl];
1046                 }
1047             }
1048         }
1049     }
1050
1051     //Interpolation is applied
1052     //b_a_L => (B-A)/L... (X-A)/l = (B-A)/L :: => X = (((B-A)/L)*l + A)
1053     for( int row=1; row<image->height-1; row++ ){
1054         uchar *ptr = ( uchar* )( image->imageData );
1055         for( int col=0; col<image->width; col++ ){
1056             for( int chl=0; chl<image->nChannels; chl++ ){
1057                 ptr[ row*image->widthStep + col*image->nChannels + chl ] = (
1058 image->nChannels + chl ] )*(
1059                 row/( float )image->height ) + ptr[ col*image->nChannels + chl ];
1060             }
1061         }
1062     }
1063 }

```

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 panel, 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

<code>filename</code>	: 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`.

```

1122                                     {
1123     int row, col;
1124     cvPutText( pnl, "Step : ", cvPoint( 3, 60 ), &font, black );
1125     cvPutText( pnl, "File : ", cvPoint( 3, 140 ), &font, black );
1126     cvPutText( pnl, filename, cvPoint( 65, 140 ), &font, black );
1127     cvPutText( pnl, "Control Pannel", cvPoint( 3, 15 ), &font_bold_italic,
black );
1128     cvPutText( pnl, "FPS : ", cvPoint( 700, 100 ), &font, black );
1129     cvPutText( pnl, "Current Frame : ", cvPoint( 3, 100 ), &font, black );
1130     cvPutText( pnl, "Total Frames : ", cvPoint( 300, 100 ), &font, black );
1131     cvPutText( pnl, "FOURCC : ", cvPoint( 668, 60 ), &font, black );
1132     cvPutText( pnl, "Status : ", cvPoint( 325, 30 ), &font, black );
1133     //Current Frame field
1134     row = 88;
1135     col = 150;
1136     cur_frame_no = cvCreateImageHeader( cvSize( 120, 18), IPL_DEPTH_8U, 3 );
1137     cur_frame_no->origin = pnl->origin;
1138     cur_frame_no->widthStep = pnl->widthStep;
1139     cur_frame_no->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1140     resetField( cur_frame_no, STATIC_TEXT );
1141     //number of frames field
1142     row = 88;
1143     col = 430;
1144     numFrames = cvCreateImageHeader( cvSize( 120, 18), IPL_DEPTH_8U, 3 );
1145     numFrames->origin = pnl->origin;
1146     numFrames->widthStep = pnl->widthStep;
1147     numFrames->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1148     resetField( numFrames, STATIC_TEXT );
1149     //Step field
1150     row = 48;
1151     col = 65;
1152     step_edit = cvCreateImageHeader( cvSize( 50, 18), IPL_DEPTH_8U, 3 );
1153     step_edit->origin = pnl->origin;
1154     step_edit->widthStep = pnl->widthStep;
1155     step_edit->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1156     resetField( step_edit, EDIT_TEXT );
1157     step_edit_area.x1 = col;
1158     step_edit_area.x2 = col + step_edit->width;
1159     step_edit_area.y1 = p_height - ctrl_pnl_height + row;

```



```

1160     step_edit_area.y2 = p_height - ctrl_pnl_height +
step_edit->height + row;
1161     sprintf( line, "%d", step_val );
1162     cvPutText( step_edit, line, cvPoint( 3, step_edit->height - 4 ), &
font, black );
1163     //FPS field
1164     row = 88;
1165     col = 755;
1166     fps_edit = cvCreateImageHeader( cvSize( 50, 18), IPL_DEPTH_8U, 3 );
1167     fps_edit->origin = pnl->origin;
1168     fps_edit->widthStep = pnl->widthStep;
1169     fps_edit->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1170     resetField( fps_edit, STATIC_TEXT );
1171     fps_edit_area.x1 = col;
1172     fps_edit_area.x2 = col + fps_edit->width;
1173     fps_edit_area.y1 = p_height - ctrl_pnl_height + row;
1174     fps_edit_area.y2 = p_height - ctrl_pnl_height +
fps_edit->height + row;
1175     //FOURCC field
1176     row = 48;
1177     col = 755;
1178     four_cc_edit = cvCreateImageHeader( cvSize( 50, 22), IPL_DEPTH_8U, 3 );
1179     four_cc_edit->origin = pnl->origin;
1180     four_cc_edit->widthStep = pnl->widthStep;
1181     four_cc_edit->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1182     resetField( four_cc_edit, STATIC_TEXT );
1183     four_cc_edit_area.x1 = col;
1184     four_cc_edit_area.x2 = col + four_cc_edit->width;
1185     four_cc_edit_area.y1 = p_height - ctrl_pnl_height + row;
1186     four_cc_edit_area.y2 = p_height - ctrl_pnl_height +
four_cc_edit->height + row;
1187     //Play/Pause button
1188     row = 48;
1189     col = 350;
1190     play_pause_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
1191     play_pause_btn->origin = pnl->origin;
1192     play_pause_btn->widthStep = pnl->widthStep;
1193     play_pause_btn->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1194     getButton( play_pause_btn, PLAY_BTN,
BTN_ACTIVE );
1195     play_pause_btn_area.x1 = col;
1196     play_pause_btn_area.x2 = col + play_pause_btn->width;
1197     play_pause_btn_area.y1 = p_height -
ctrl_pnl_height + row;
1198     play_pause_btn_area.y2 = p_height -
ctrl_pnl_height + play_pause_btn->height + row;
1199     //Stop button
1200     row = 48;
1201     col = 415;
1202     stop_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
1203     stop_btn->origin = pnl->origin;
1204     stop_btn->widthStep = pnl->widthStep;
1205     stop_btn->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1206     getButton( stop_btn, STOP_BTN, BTN_ACTIVE );
1207     stop_btn_area.x1 = col;
1208     stop_btn_area.x2 = col + stop_btn->width;
1209     stop_btn_area.y1 = p_height - ctrl_pnl_height + row;
1210     stop_btn_area.y2 = p_height - ctrl_pnl_height +
stop_btn->height + row;
1211     //Stepup button
1212     row = 48;
1213     col = 480;
1214     stepup_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
1215     stepup_btn->origin = pnl->origin;
1216     stepup_btn->widthStep = pnl->widthStep;
1217     stepup_btn->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1218     getButton( stepup_btn, STEPUP_BTN, BTN_ACTIVE );
1219     stepup_btn_area.x1 = col;
1220     stepup_btn_area.x2 = col + stepup_btn->width;
1221     stepup_btn_area.y1 = p_height - ctrl_pnl_height + row;
1222     stepup_btn_area.y2 = p_height - ctrl_pnl_height +
stepup_btn->height + row;
1223     //Stepdown button
1224     row = 48;
1225     col = 285;
1226     stepdown_btn = cvCreateImageHeader( cvSize( 60, 18), IPL_DEPTH_8U, 3 );
1227     stepdown_btn->origin = pnl->origin;
1228     stepdown_btn->widthStep = pnl->widthStep;
1229     stepdown_btn->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1230     getButton( stepdown_btn, STEPDOWN_BTN,
BTN_ACTIVE );

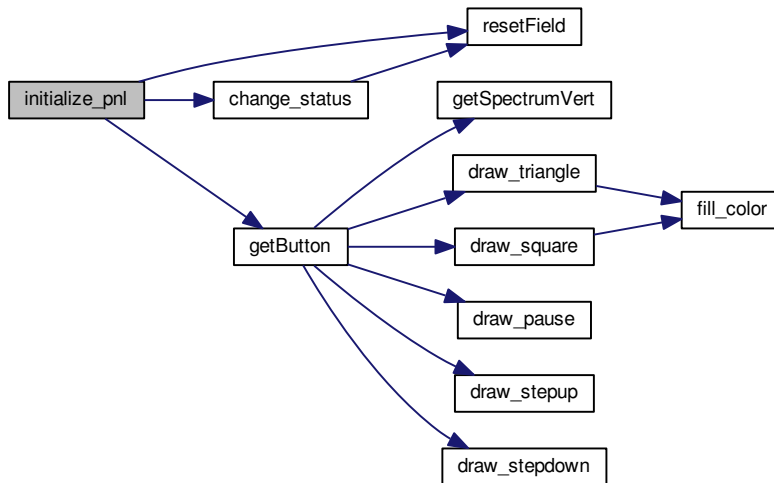
```

```

1231     stepdown_btn_area.x1 = col;
1232     stepdown_btn_area.x2 = col + stepdown_btn->width;
1233     stepdown_btn_area.y1 = p_height - ctrl_pnl_height + row;
1234     stepdown_btn_area.y2 = p_height - ctrl_pnl_height +
stepdown_btn->height + row;
1235     //Status Field
1236     row = 18;
1237     col = 395;
1238     status_edit = cvCreateImageHeader( cvSize( 130, 22), IPL_DEPTH_8U, 3 );
1239     status_edit->origin = pnl->origin;
1240     status_edit->widthStep = pnl->widthStep;
1241     status_edit->imageData = pnl->imageData + row*pnl->widthStep + col*
pnl->nChannels;
1242     resetField( status_edit, STATIC_TEXT );
1243     status_edit_area.x1 = col;
1244     status_edit_area.x2 = col + status_edit->width;
1245     status_edit_area.y1 = p_height - ctrl_pnl_height + row;
1246     status_edit_area.y2 = p_height - ctrl_pnl_height +
status_edit->height + row;
1247     sprintf( status_line, "Stopped" );
1248     change_status();
1249 }

```

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-panel sub-image is assigned as a part of the main *player* image.

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

Above the control-panel, a sub-image is assigned to be a slider. OpenCV has an inbuilt function `cvCreateTrackbar()` to create a slider. But the disadvantage with this function is, the slider is placed at either at the top or the bottom 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 achieving 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 Panel*. To access the video properites `cvGetCaptureProperty()` 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 views.

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

Parameters

<code>argv[1]</code>	: Video file path
----------------------	-------------------

Return values

<code>0</code>	Exit without any problem.
<code>1</code>	Early exit with due to some error.

Definition at line 505 of file video_player.c.

```

505                                     {
506
507     //Initialize the font
510     cvInitFont( &font, font_face, hscale, vscale, shear,
    thickness, line_type );
511     cvInitFont( &font_italic, font_face_italic,
    hscale, vscale, shear, thickness, line_type );
512     cvInitFont( &font_bold, font_face, hscale, vscale,
    shear, thickness+1, line_type );
513     cvInitFont( &font_bold_italic, font_face_italic,
    hscale, vscale, shear, thickness+1, line_type );

```

```

514
515 //Create the player image
518 player = cvCreateImage( cvSize( p_width, p_height ), IPL_DEPTH_8U, 3 );
519
520 //Create Control Pannel
523 pnl = cvCreateImageHeader( cvSize( p_width, ctrl_pnl_height ), IPL_DEPTH_8U, 3
);
524 pnl->origin = player->origin;
525 pnl->widthStep = player->widthStep;
526 pnl->imageData = player->imageData + ( p_height -
ctrl_pnl_height ) * player->widthStep;
527 for( int row=0; row<pnl->height; row++){
528     uchar* ptr = ( uchar* )( pnl->imageData + row*pnl->widthStep );
529     for( int col=0; col<pnl->width; col++){
530         ptr[ col*pnl->nChannels + 0 ] = 226;
531         ptr[ col*pnl->nChannels + 1 ] = 235;
532         ptr[ col*pnl->nChannels + 2 ] = 240;
533     }
534 }
535 //Add text & buttons
538 initialize_pnl( argv[1] );
539
540 //create custom slider (non-opencv)
543 slider = cvCreateImageHeader( cvSize( p_width, 10 ), IPL_DEPTH_8U, 3 );
544 slider->origin = player->origin;
545 slider->widthStep = player->widthStep;
546 slider->imageData = player->imageData + ( p_height -
sldr_height - ctrl_pnl_height ) * player->widthStep;
547 for( int row=0; row<slider->height; row++){
548     uchar* ptr = ( uchar* )( slider->imageData + row*slider->widthStep );
549     for( int col=0; col<slider->width; col++){
550         ptr[ col*slider->nChannels + 0 ] = 94;
551         ptr[ col*slider->nChannels + 1 ] = 118;
552         ptr[ col*slider->nChannels + 2 ] = 254;
553     }
554 }
555 oslider = cvCloneImage( slider );
556 sldr_btn = cvCreateImage( cvSize( 15, sldr_height ), IPL_DEPTH_8U, 3 );
557 for( int row=0; row<sldr_btn->height; row++){
558     uchar* ptr = ( uchar* )( sldr_btn->imageData + row*sldr_btn->widthStep );
559     for( int col=0; col<sldr_btn->width; col++){
560         ptr[ col*slider->nChannels + 0 ] = 100;
561         ptr[ col*slider->nChannels + 1 ] = 150;
562         ptr[ col*slider->nChannels + 2 ] = 100;
563     }
564 }
565 sldr_val = cvCreateImageHeader( cvSize( sldr_btn_width,
sldr_height ), IPL_DEPTH_8U, 3 );
566 sldr_val->origin = slider->origin;
567 sldr_val->widthStep = slider->widthStep;
568 sldr_val->imageData = slider->imageData;
569 cvCopy( sldr_btn, sldr_val );
570
571 //display window
575 cvNamedWindow( "Video Player", CV_WINDOW_AUTOSIZE );
576
577 //install mouse callback
581 cvSetMouseCallback(
582     "Video Player",
583     my_mouse_callback,
584     ( void* ) NULL
);
585
586
587
588 //load the video
592 vid = cvCaptureFromFile( argv[1] );
593 //check the video
594 if( !vid ){
595     printf( "Error loading the video file. Either missing file or codec not installed\n" );
596     return( 1 );
597 }
598 frame_area = cvCreateImageHeader( cvSize( p_width,
scrn_height ), IPL_DEPTH_8U, 3 );
599 frame_area->origin = player->origin;
600 frame_area->widthStep = player->widthStep;
601 frame_area->imageData = player->imageData;
602 fps = cvGetCaptureProperty( vid, CV_CAP_PROP_FPS );
603 sldr_start = cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES );
604 fourcc_l = cvGetCaptureProperty( vid, CV_CAP_PROP_FOURCC );
605 fourcc = ( char* )( &fourcc_l );
606 sprintf( four_cc_str, "%c%c%c%c", fourcc[0], fourcc[1],
fourcc[2], fourcc[3] );
607 //printf( "FPS : %f\n", fps );
608 sldr_maxval = cvGetCaptureProperty( vid, CV_CAP_PROP_FRAME_COUNT ); //check this property
609 if( sldr_maxval < 1 ){
610     printf( "Number of frames < 1. Cannot continue...\n" );
611     return( 1 );

```

```

612     }
613     cvSetCaptureProperty(
614         vid,
615         CV_CAP_PROP_POS_FRAMES,
616         sldr_start
617     );
618     sprintf( line, "%d", sldr_maxval );
619     cvPutText( numFrames, line, cvPoint( 3, numFrames->height - 4 ), &
font, black );
620     sprintf( line, "%d", ( int )cvRound( fps ) );
621     cvPutText( fps_edit, line, cvPoint( 3, fps_edit->height - 4 ), &
font, black );
622     sprintf( line, "%d", sldr_start );
623     cvPutText( cur_frame_no, line, cvPoint( 3, cur_frame_no->height - 4 ), &
font, black );
624     sprintf( line, "%s", four_cc_str );
625     cvPutText( four_cc_edit, line, cvPoint( 3, four_cc_edit->height - 8 ), &
font, black );
626     moveSlider( sldr_start, OTHER_CALLS );
627
631     frame = cvQueryFrame( vid );
632     old_frame = cvCloneImage( frame );
633     if( !frame ){
634         printf( "Cannot load video. Missing Codec : %s\n", four_cc_str );
635         return( 1 );
636     }
637     cvShowImage( "Video Player", player );
638
642     char c;
643     int cur_frame;
644     while( 1 ){
645         if( ( c = cvWaitKey( 1000/fps ) ) == 27 ){
646             break;
647         }
648         if( !processing ){
649             if( playing ){
650                 for( int i = 0; i < ( step_val - 1 ); i++ ){
651                     cvQueryFrame( vid );
652                 }
653                 frame = cvQueryFrame( vid );
654                 if( !frame ){
655                     playing = false;
656                 }
657             } else{
658                 cvCopy( frame, old_frame );
659             }
660         }
661         //to avoid any negative value of cur_frame
662         while( 1 ){
663             if( ( cur_frame = ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES ) ) >= 0 ){
664                 break;
665             }
666             //for some unknown reason cvQueryFrame was needed to be called twice to get to the desired
frame.
667             frame = cvQueryFrame( vid );
668             cvCopy( frame, old_frame );
669         }
670         //defines the task to be carried out when editing a text-field
671         if( typing_step ){
672             type_step( c, cur_frame );
673         }
674         //this takes care if for some reason the cur_frame overshoots the sldr_maxval.
675         if( cur_frame == ( sldr_maxval-1 ) ){
676             getButton( play_pause_btn, PLAY_BTN,
BTN_ACTIVE );
677             sprintf( status_line, "End reached" );
678             change_status();
679         }
680         cvResize( old_frame, frame_area );
681         //printf( "Current frame : %d\n", cur_frame );
682         moveSlider( cur_frame, OTHER_CALLS );
683     }
684     cvShowImage( "Video Player", player );
685 }
686
690 //destroy window
691 cvDestroyWindow( "Video Player" );
692
693 //Release image
694 cvReleaseImageHeader( &stepdown_btn );
695 cvReleaseImageHeader( &stepup_btn );
696 cvReleaseImageHeader( &stop_btn );
697 cvReleaseImageHeader( &play_pause_btn );
698 cvReleaseImageHeader( &step_edit );
699 cvReleaseImageHeader( &four_cc_edit );
700 cvReleaseImageHeader( &fps_edit );
701 cvReleaseImageHeader( &numFrames );

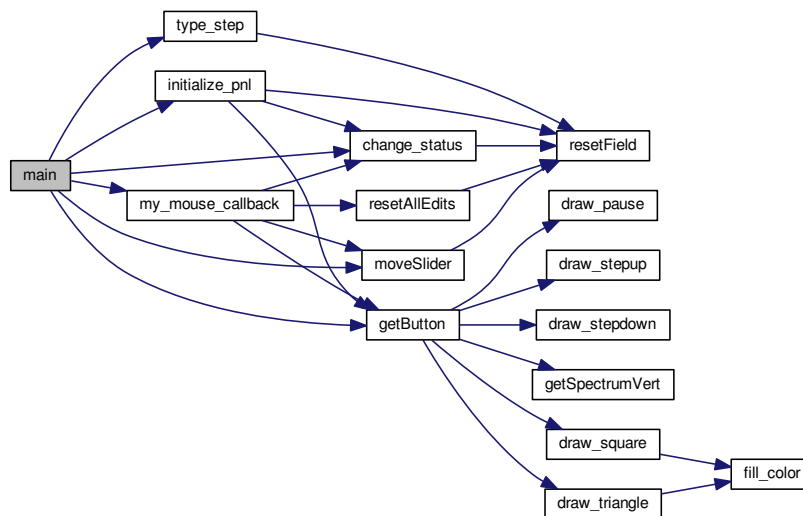
```

```

702     cvReleaseImageHeader( &cur_frame_no );
703     cvReleaseImageHeader( &pnl );
704     cvReleaseImageHeader( &sldr_val );
705     cvReleaseImageHeader( &slider );
706     cvReleaseImageHeader( &frame_area );
707     cvReleaseImage( &old_frame );
708     cvReleaseImage( &sldr_btn );
709     cvReleaseImage( &oslider );
710     cvReleaseImage( &player );
711
712     //Release the video
713     cvReleaseCapture( &vid );
714
715     return( 0 );
721 }

```

Here is the call graph for this function:



5.1.3.13 int moveSlider (int pos, int call_from)

Custom 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 integral 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

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

Returns

`frame_val`: The current frame number.

See Also

[resetField\(\)](#), [cvPutText\(\)](#), [cvCopy\(\)](#)

Definition at line 729 of file `video_player.c`.

```

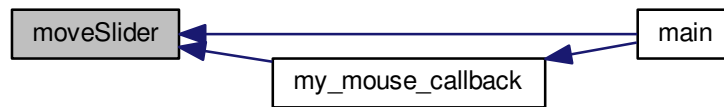
729                                     {
730     int frame_val;
731     //Scaling to obtain the current frame number
732     float scale = ( sldr_maxval )/( float )( p_width );
733     //printf( "Pos : %d\tScale : %f\n", pos, scale );
734     if( call_from == MOUSE_CALLBACK ){
735         frame_val = cvCeil( scale*pos );
736     }
737     if( call_from == OTHER_CALLS ){
738         frame_val = pos;
739     }
740     //Scaling to set the slider button at an appropriate location between 0 and (p_width - sldr_btn_width)
741     scale = ( p_width - sldr_btn_width )/( float )(
742     sldr_maxval );
743     //printf( "Frame slider : %d\n", frame_val );
744     int new_pos = cvCeil( scale*frame_val );
745     //frame_val should be an integral multiple of step_val
746     if( frame_val%step_val != 0 ){
747         frame_val = step_val*( ( int )frame_val/( int )step_val );
748     }
749     resetField( cur_frame_no, STATIC_TEXT );
750     sprintf( line, "%d", frame_val );
751     cvPutText( cur_frame_no, line, cvPoint( 3, cur_frame_no->height - 4 ), &font,
752     black );
753     cvCopy( oslider, slider );
754     sldr_val->imageData = slider->imageData + new_pos*slider->nChannels;
755     cvCopy( sldr_btn, sldr_val );
756     return( frame_val );
757 }

```

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 explanation 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. Appropriate 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

<i>event</i>	
<i>x</i>	
<i>y</i>	
<i>flags</i>	
<i>param</i>	

Definition at line 784 of file video_player.c.

```

784                                     {
785     IplImage* image = ( IplImage* )param;
786     switch( event ){
790         case CV_EVENT_MOUSEMOVE: {
791             if( sldr_moving ){
792                 // mouse on slider
793                 if( ( y > scrn_height ) && ( y <= scrn_height +
sldr_height ) ){
794                     int cur_frame = moveSlider( x, MOUSE_CALLBACK );
795                     if( vid ){
796                         cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )( cur_frame-1 ) );
797                         cvQueryFrame( vid );
798                         cvCopy( cvQueryFrame( vid ), old_frame );
799                     }
800                 }
  
```



```

801     }
802 }
803 break;
807 case CV_EVENT_LBUTTONDOWN: {
808     sldr_moving = true;
809     resetAllEdits();
810     // mouse on slider
811     if( ( y > scrn_height ) && ( y <= scrn_height +
sldr_height ) ){
812         int cur_frame = moveSlider( x, MOUSE_CALLBACK );
813         if( vid ){
814             cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )( cur_frame-1 +
step_val -1 ) );
815             cvQueryFrame( vid );
816             cvCopy( cvQueryFrame( vid ), old_frame );
817             //printf( "Before val : %f\n", cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES ) );
818         }
819         if( !playing ){
820             sprintf( status_line, "Slider moved" );
821             change_status();
822         }
823     }
824     // mouse on play/pause button
825     if(
826         ( y > play_pause_btn_area.y1 ) &&
827         ( y <= play_pause_btn_area.y2 ) &&
828         ( x > play_pause_btn_area.x1 ) &&
829         ( x <= play_pause_btn_area.x2 )
830     ){
831         //printf( "Frame val : %d\n", ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES ) );
832         if( playing ){
833             playing = false;
834             getButton( play_pause_btn, PLAY_BTN,
BTN_ACTIVE );
835             sprintf( status_line, "Paused" );
836             change_status();
837         }
838         else{
839             playing = true;
840             getButton( play_pause_btn, PAUSE_BTN,
BTN_ACTIVE );
841             sprintf( status_line, "Playing" );
842             change_status();
843         }
844     }
845     // mouse on stop button
846     if(
847         ( y > stop_btn_area.y1 ) &&
848         ( y <= stop_btn_area.y2 ) &&
849         ( x > stop_btn_area.x1 ) &&
850         ( x <= stop_btn_area.x2 )
851     ){
852         playing = false;
853         moveSlider( sldr_start, OTHER_CALLS );
854         if( vid ){
855             cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )(
sldr_start-1 ) );
856             cvQueryFrame( vid );
857             cvCopy( cvQueryFrame( vid ), old_frame );
858         }
859         getButton( play_pause_btn, PLAY_BTN,
BTN_ACTIVE );
860         sprintf( status_line, "Stopped" );
861         change_status();
862     }
863     // mouse on stepup button
864     if(
865         ( y > stepup_btn_area.y1 ) &&
866         ( y <= stepup_btn_area.y2 ) &&
867         ( x > stepup_btn_area.x1 ) &&
868         ( x <= stepup_btn_area.x2 )
869     ){
870         int cur_frame = ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES );
871         //printf( "Frame val : %d\n", cur_frame );
872         if( cur_frame + 1 + step_val - 1 < sldr_maxval ){
873             for( int i=0; i < ( step_val - 1 ); i++){
874                 cvQueryFrame( vid );
875             }
876             frame = cvQueryFrame( vid );
877             if( frame ){
878                 cvCopy( frame, old_frame );
879             }
880         }
881         if( !playing ){
882             sprintf( status_line, "Stepped Up" );
883             change_status();
884         }

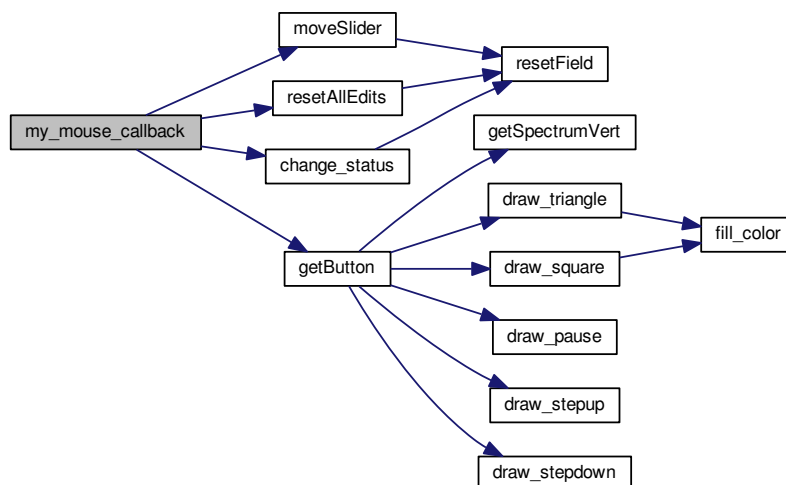
```

```

885         //printf( "Stepup pressed \n" );
886     }
887     // mouse on stepdown button
888     if(
889         ( y > stepdown_btn_area.y1 ) &&
890         ( y <= stepdown_btn_area.y2 ) &&
891         ( x > stepdown_btn_area.x1 ) &&
892         ( x <= stepdown_btn_area.x2 )
893     ){
894         processing = true;
895         int cur_frame = ( int )cvGetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES );
896         //printf( "Frame val : %d\n", cur_frame );
897         if( cur_frame - 1 - ( step_val - 1 ) >= sldr_start ){
898             moveSlider( ( cur_frame - 1 - ( step_val - 1 ) ),
OTHER_CALLS );
899             cvSetCaptureProperty( vid, CV_CAP_PROP_POS_FRAMES, ( double )( cur_frame - 1 - (
step_val - 1 ) ) );
900             cvQueryFrame( vid );
901             cvCopy( cvQueryFrame( vid ), old_frame );
902             //printf( "New Frame val : %d\n", ( int )cvGetCaptureProperty( vid,
CV_CAP_PROP_POS_FRAMES ) );
903         }
904         if( !playing ){
905             sprintf( status_line, "Stepped Down" );
906             change_status();
907         }
908         processing = false;
909         //printf( "Stepdown pressed \n" );
910     }
911     // mouse on step_edit field
912     if(
913         ( y > step_edit_area.y1 ) &&
914         ( y <= step_edit_area.y2 ) &&
915         ( x > step_edit_area.x1 ) &&
916         ( x <= step_edit_area.x2 )
917     ){
918         sprintf( edit_text, "" );
919         typing_step = true;
920     }
921 }
922 break;
926 case CV_EVENT_LBUTTONDOWN: {
927     sldr_moving = false;
928 }
929 break;
930 }
939 }

```

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.

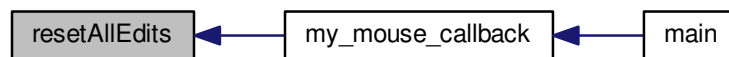
```

1466     {
1467     resetField( step_edit, EDIT_TEXT );
1468     sprintf( edit_text, "%d", step_val );
1469     cvPutText( step_edit, edit_text, cvPoint( 3, step_edit->height - 4 ), &
font, black );
1470     typing_step = false;
1471 }
  
```

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 respective 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

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

See Also

[IplImage](#)

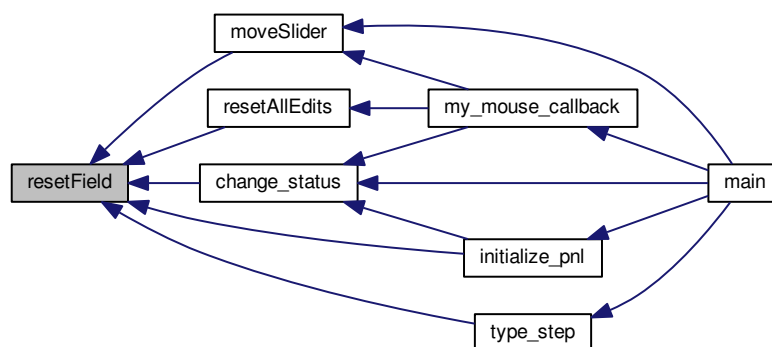
Definition at line 952 of file video_player.c.

```

952                                     {
953     if( text_type == STATIC_TEXT ){
954         for( int row=0; row<image->height; row++ ){
955             uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
956             for( int col=0; col<image->width; col++ ){
957                 ptr[ col*image->nChannels + 0 ] = 226;
958                 ptr[ col*image->nChannels + 1 ] = 235;
959                 ptr[ col*image->nChannels + 2 ] = 240;
960             }
961         }
962     }
963     else{
964         for( int row=0; row<image->height; row++ ){
965             uchar *ptr = ( uchar* )( image->imageData + row*image->widthStep );
966             for( int col=0; col<image->width; col++ ){
967                 if( row==0 || row==image->height-1 || col==0 || col==image->width-1 ){
968                     ptr[ col*image->nChannels + 0 ] = 0;
969                     ptr[ col*image->nChannels + 1 ] = 0;
970                     ptr[ col*image->nChannels + 2 ] = 0;
971                 }
972                 else
973                 {
974                     ptr[ col*image->nChannels + 0 ] = 255;
975                     ptr[ col*image->nChannels + 1 ] = 255;
976                     ptr[ col*image->nChannels + 2 ] = 255;
977                 }
978             }
979         }
980     }
981 }

```

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.

```

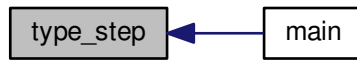
1411
1412     resetField( step_edit, EDIT_TEXT );
1413     char temp_text[ 20 ];
1414     int cur_frame;
1415     sprintf( temp_text, "" );
1416     if( blinking ){
1417         if( blink_count<blink_max ){
1418             blink_count++;
1419         }
1420         else{
1421             blinking = false;
1422             blink_char = ' ';
1423             blink_count = 0;
1424         }
1425         //printf( "Blinking...\n" );
1426     }
1427     else{
1428         if( blink_count<blink_max ){
1429             blink_count++;
1430         }
1431         else{
1432             blinking = true;
1433             blink_char = '|';
1434             blink_count = 0;
1435         }
1436         //printf( "Not blinking...\n" );
1437     }
1438     //valid number
1439     if( c>=48 && c<=57 ){
1440         sprintf( temp_text, "%s%c", edit_text, c );
1441         if( ( frame_val + atoi( temp_text ) )>=0 && ( frame_val + atoi( temp_text ) )<=
sldr_maxval && ( atoi( temp_text )!=0 )){
1442             sprintf( edit_text, "%s", temp_text );
1443         }
1444     }
1445     //backspace
1446     if( c==8 ){
1447         if( strcmp( edit_text, "" )!=0 ){
1448             for( int count=0; count<( strlen( edit_text )-1 ); count++ ){
1449                 sprintf( temp_text, "%s%c", temp_text, edit_text[ count ] );
1450             }
1451             sprintf( edit_text, "%s", temp_text );
1452         }
1453     }
1454     sprintf( temp_text, "%s%c", edit_text, blink_char );
1455     cvPutText( step_edit, temp_text, cvPoint( 3, step_edit->height - 4 ), &
font, black );
1456     if( c==10 ){
1457         resetField( step_edit, EDIT_TEXT );
1458         cvPutText( step_edit, edit_text, cvPoint( 3, step_edit->height - 4 ), &
font, black );
1459         step_val = atoi( edit_text );
1460         //printf( "Step : %d\n", step );
1461         typing_step = false;
1462     }
1463 }

```

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 toggle 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 toggle the blinker character *blink_char*. Whenever this counter crosses *blink_max*, the *blink_char* is toggled.

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 IpImage* four_cc_edit

Pointer to FOUR_CC static-text.

Points to the sub-image showing FOUR_CC static text.

See Also

[Ip1Image](#), [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 IpImage* 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, toggled 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-panel sub-image.

The control-panel 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-Panel 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 `Ip1Image* stepdown_btn`

Pointer to `step_down` button area.

Points to the sub-image having the `step_down` button.

See Also

[Ip1Image](#), [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 `Ip1Image* stepup_btn`

Pointer to `step_up` button area.

Points to the sub-image having the `step_up` button.

See Also

[Ip1Image](#), [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 `Ip1Image* stop_btn`

Pointer to stop button area.

Points to the sub-image having the stop button.

See Also

[Ip1Image](#), [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 violet = 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

- BTN_ACTIVE
 - video_player.c, [14](#)
- BTN_INACTIVE
 - video_player.c, [14](#)
- black
 - video_player.c, [41](#)
- blink_char
 - video_player.c, [41](#)
- blink_count
 - video_player.c, [41](#)
- blink_max
 - video_player.c, [41](#)
- blinking
 - video_player.c, [41](#)
- blue
 - video_player.c, [41](#)
- brown
 - video_player.c, [41](#)

- change_status
 - video_player.c, [17](#)
- ctrl_pnl_height
 - video_player.c, [14](#)
- cur_frame_no
 - video_player.c, [42](#)

- draw_pause
 - video_player.c, [17](#)
- draw_square
 - video_player.c, [18](#)
- draw_stepdown
 - video_player.c, [19](#)
- draw_stepup
 - video_player.c, [20](#)
- draw_triangle
 - video_player.c, [20](#)

- EDIT_CALLS
 - video_player.c, [14](#)
- EDIT_TEXT
 - video_player.c, [14](#)
- edit_text
 - video_player.c, [42](#)

- Field_Area, [7](#)
 - x1, [7](#)
 - x2, [7](#)
 - y1, [8](#)
 - y2, [8](#)
- fill_color
 - video_player.c, [21](#)
- font
 - video_player.c, [42](#)
- font_bold
 - video_player.c, [42](#)
- font_bold_italic
 - video_player.c, [42](#)
- font_face
 - video_player.c, [42](#)
- font_face_italic
 - video_player.c, [42](#)
- font_italic
 - video_player.c, [42](#)
- four_cc_edit
 - video_player.c, [43](#)
- four_cc_edit_area
 - video_player.c, [43](#)
- four_cc_str
 - video_player.c, [43](#)
- fourcc
 - video_player.c, [43](#)
- fourcc_l
 - video_player.c, [43](#)
- fps
 - video_player.c, [43](#)
- fps_edit
 - video_player.c, [43](#)
- fps_edit_area
 - video_player.c, [44](#)
- frame
 - video_player.c, [44](#)
- frame_area
 - video_player.c, [44](#)

- getButton
 - video_player.c, [22](#)
- getSpectrumHorz
 - video_player.c, [23](#)
- getSpectrumVert
 - video_player.c, [24](#)
- gray
 - video_player.c, [44](#)
- green
 - video_player.c, [44](#)

- hscale
 - video_player.c, [45](#)

- initialize_pnl
 - video_player.c, [25](#)

- light_yellow
 - video_player.c, [45](#)
- line
 - video_player.c, [45](#)
- line_type
 - video_player.c, [45](#)
- MOUSE_CALLBACK
 - video_player.c, [14](#)
- main
 - video_player.c, [28](#)
- moveSlider
 - video_player.c, [32](#)
- my_mouse_callback
 - video_player.c, [34](#)
- numFrames
 - video_player.c, [45](#)
- OTHER_CALLS
 - video_player.c, [15](#)
- old_frame
 - video_player.c, [45](#)
- orange
 - video_player.c, [45](#)
- oslider
 - video_player.c, [46](#)
- p_height
 - video_player.c, [15](#)
- p_width
 - video_player.c, [15](#)
- PAUSE_BTN
 - video_player.c, [15](#)
- PLAY_BTN
 - video_player.c, [15](#)
- play_pause_btn
 - video_player.c, [46](#)
- play_pause_btn_area
 - video_player.c, [46](#)
- player
 - video_player.c, [46](#)
- playing
 - video_player.c, [46](#)
- pnl
 - video_player.c, [47](#)
- processing
 - video_player.c, [47](#)
- red
 - video_player.c, [47](#)
- resetAllEdits
 - video_player.c, [37](#)
- resetField
 - video_player.c, [37](#)
- STATIC_TEXT
 - video_player.c, [16](#)
- STEPDOWN_BTN
 - video_player.c, [16](#)
- STEPUP_BTN
 - video_player.c, [16](#)
- STOP_BTN
 - video_player.c, [16](#)
- scrn_height
 - video_player.c, [15](#)
- shear
 - video_player.c, [47](#)
- sldr_btn
 - video_player.c, [47](#)
- sldr_btn_width
 - video_player.c, [16](#)
- sldr_height
 - video_player.c, [16](#)
- sldr_maxval
 - video_player.c, [48](#)
- sldr_moving
 - video_player.c, [48](#)
- sldr_start
 - video_player.c, [48](#)
- sldr_val
 - video_player.c, [48](#)
- slider
 - video_player.c, [48](#)
- status_edit
 - video_player.c, [49](#)
- status_edit_area
 - video_player.c, [49](#)
- status_line
 - video_player.c, [49](#)
- step_edit
 - video_player.c, [49](#)
- step_edit_area
 - video_player.c, [49](#)
- step_val
 - video_player.c, [49](#)
- stepdown_btn
 - video_player.c, [49](#)
- stepdown_btn_area
 - video_player.c, [50](#)
- stepup_btn
 - video_player.c, [50](#)
- stepup_btn_area
 - video_player.c, [50](#)
- stop_btn
 - video_player.c, [50](#)
- stop_btn_area
 - video_player.c, [50](#)
- thickness
 - video_player.c, [50](#)
- type_step
 - video_player.c, [39](#)
- typing_step
 - video_player.c, [50](#)
- vid
 - video_player.c, [51](#)
- video_player.c, [9](#)

- BTN_ACTIVE, 14
- BTN_INACTIVE, 14
- black, 41
- blink_char, 41
- blink_count, 41
- blink_max, 41
- blinking, 41
- blue, 41
- brown, 41
- change_status, 17
- ctrl_pnl_height, 14
- cur_frame_no, 42
- draw_pause, 17
- draw_square, 18
- draw_stepdown, 19
- draw_stepup, 20
- draw_triangle, 20
- EDIT_CALLS, 14
- EDIT_TEXT, 14
- edit_text, 42
- fill_color, 21
- font, 42
- font_bold, 42
- font_bold_italic, 42
- font_face, 42
- font_face_italic, 42
- font_italic, 42
- four_cc_edit, 43
- four_cc_edit_area, 43
- four_cc_str, 43
- fourcc, 43
- fourcc_l, 43
- fps, 43
- fps_edit, 43
- fps_edit_area, 44
- frame, 44
- frame_area, 44
- getButton, 22
- getSpectrumHorz, 23
- getSpectrumVert, 24
- gray, 44
- green, 44
- hscale, 45
- initialize_pnl, 25
- light_yellow, 45
- line, 45
- line_type, 45
- MOUSE_CALLBACK, 14
- main, 28
- moveSlider, 32
- my_mouse_callback, 34
- numFrames, 45
- OTHER_CALLS, 15
- old_frame, 45
- orange, 45
- oslider, 46
- p_height, 15
- p_width, 15
- PAUSE_BTN, 15
- PLAY_BTN, 15
- play_pause_btn, 46
- play_pause_btn_area, 46
- player, 46
- playing, 46
- pnl, 47
- processing, 47
- red, 47
- resetAllEdits, 37
- resetField, 37
- STATIC_TEXT, 16
- STEPDOWN_BTN, 16
- STEPUP_BTN, 16
- STOP_BTN, 16
- scrn_height, 15
- shear, 47
- sldr_btn, 47
- sldr_btn_width, 16
- sldr_height, 16
- sldr_maxval, 48
- sldr_moving, 48
- sldr_start, 48
- sldr_val, 48
- slider, 48
- status_edit, 49
- status_edit_area, 49
- status_line, 49
- step_edit, 49
- step_edit_area, 49
- step_val, 49
- stepdown_btn, 49
- stepdown_btn_area, 50
- stepup_btn, 50
- stepup_btn_area, 50
- stop_btn, 50
- stop_btn_area, 50
- thickness, 50
- type_step, 39
- typing_step, 50
- vid, 51
- voilet, 51
- vscale, 51
- white, 51
- yellow, 51
- voilet
 - video_player.c, 51
- vscale
 - video_player.c, 51
- white
 - video_player.c, 51
- x1
 - Field_Area, 7
- x2
 - Field_Area, 7
- y1

Field_Area, [8](#)
y2
Field_Area, [8](#)
yellow
video_player.c, [51](#)