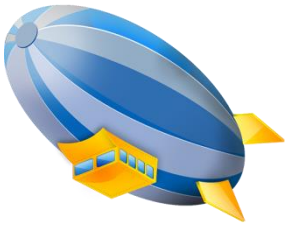


Blimp

OSD development tool

By ANALOG DEVICES

ADV7625 Framework User Guide

Preliminary 1.1

Advantiv
Advanced Television Solutions
by Analog Devices

ANALOG DEVICES

Blimp Framework User Manual for 3.9Rel
December 2015

Preface

Product Overview

The Blimp tool is used to:

- Define through its graphical interface and component properties the appearance of OSD menus, submenus, and items
- Define how the navigation within the OSD works in response to user input
- Call the appropriate system level function based on user interaction with the OSD
- Compile and generate the ANSI-C files needed for OSD system level integration

Purpose of This Manual

The purpose of the *ADV7625 Blimp Framework User Guide* document is to provide a detailed description of each component and functionality in Blimp specific to the ADV7625 framework.

Intended Audience

The primary audience for this manual is OSD designers and software developers creating OSD designs using the Blimp tool.

Manual Contents

The manual consists of:

- Chapter 1, Osd framework overview

Provides basic information about the OSD firmware.

- Chapter 2,OSD API

Provides information about pages in Blimp.

- Chapter 3, Using OSD Components

Describes the OSD components, properties, methods and events.

- Chapter 4, Project Settings
  Provides information about project settings.
- Chapter 5, Resolution configuration
  Provides information and Configuration for all supported resolutions.

Technical or Customer Support

You can reach Analog Devices, Inc. Customer Support in the following ways:

- Visit the Audio/Video Products Web site at http://www.analog.com/en/audiovideo-products/analoghdmidvi-interfaces/products/index.html
- Post questions at http://ez.analog.com/community/video
- Phone questions to **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:

Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
USA

Product Information

Product information can be obtained from the Analog Devices Web site and other Web sources.

Analog Devices Web Site

The Analog Devices Web site,www.analog.com, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to http://www.analog.com/processors/technical_library. The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals.MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

Engineer Zone

Engineer Zone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Copyright Information

Disclaimer

Analog Devices, Inc. (ADI) reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

The information contained in this document is proprietary of ADI. This document must not be made available to anybody other than the intended recipient without the written permission of ADI.

The content of this document is believed to be correct. If any errors are found within this document or if clarification is needed, contact the video community on EZ forum (http://ez.analog.com/community/video).

Social Networking Web Sites

You can now follow Analog Devices processor development on Twitter and LinkedIn. To access:

Twitter:http://twitter.com/ADIsharcandhttp://twitter.com/blackfin
LinkedIn: Network with the LinkedIn group: http://www.linkedin.com

Trademark and Service Mark Notice

The Analog Devices logo is a registered trademark of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Analog Devices' Trademarks and Service Marks may not be used without the express written consent of Analog Devices, such consent only to be provided in a separate written agreement signed by Analog Devices. Subject to the foregoing, such Trademarks and Service Marks must be used according to Analog Devices' Trademark Usage guidelines. Any licensee wishing to use Analog Devices' Trademarks and Service Marks must obtain and follow these guidelines for the specific marks at issue.

Table of contents

1 Osd framework overview

This chapter provides basic information about the OSD firmware.

The following topics are covered:

1.1 Introduction

The firmware is located between the Blimp OSD tool and the OSD and hardware, receiving files generated by the Blimp OSD tool and the controlling hardware to draw a user customized graphical user interface (GUI). Figure 1 provides a simplified block diagram of the low level firmware.

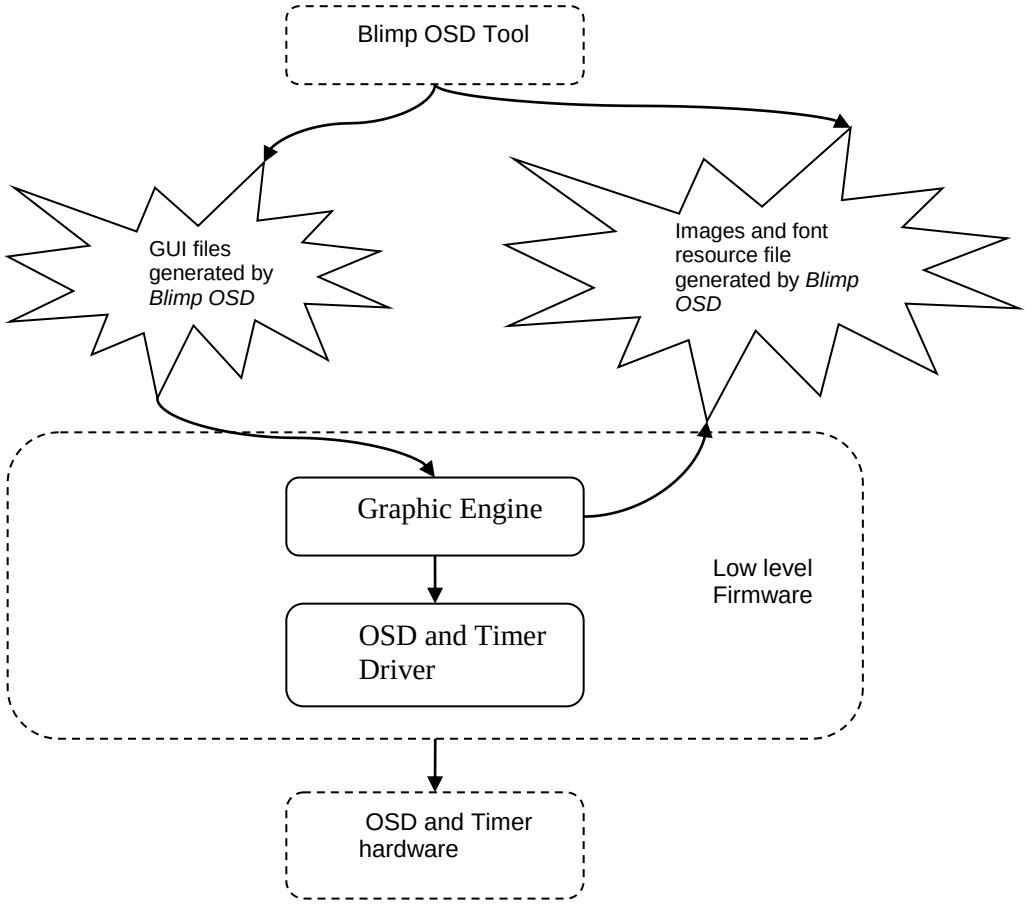


Figure 1.Block Diagram of Low Level Firmware

In the process of initializing the OSD engine, just one API is called, inside which the firmware registers components, initializes hardware and calls the GUI source files generated by the Blimp OSD tool to create GUI and map messages/events. As the firmware is message/event driven, for any message/event sent to the firmware current focus window (including keypad input and infrared remote controller input) the GUI will operate the same as the customized in Blimp OSD tool.

1.1.1 Graphic Engine

1.1.2 Introduction

The graphic engine is a powerful but compact package. It implements all components with same interfaces as Blimp OSD tool, supports message/event mapping and has animation capability. Besides, the components and messages are expandable, which means user could create their specific components and add customized messages/events.

Table 1 lists the components that are already implemented in the firmware.

Table 1Implemented OSD Components in Firmware

OSD Component	Description
OSD_LABEL	Used to display text
OSD_LISTBOX	Displays a list of user-selectable options
OSD_IMAGE	Container for an image animation using Ibox
OSD_TBOXIMAGE	Container for an image or animation using Tbox
OSD_PROGRESSBAR	Displays a progress bar
OSD_BOX	Display the filled box with border
OSD_HISTOGRAM	Displays a graphic equalizer

1.1.3 Components and Drawing Units

The relationship between components and drawing units is shown inTable 2.

Table 2Map between Component and drawing units

OSD Component	Drawing Units
Label	n Tbox
Listbox	n Tbox
Image	n Ibox
TboxImage	n Tbox
Progressbar	2 or 3 Fbox
Box	1 Fbox
Histogram	n Fbox

1.1.4 Hardware Drawing Units

The drawing Units are used by Blimp OSD components to drive the graphic components from the hardware. A basic understanding of these drawing units is necessary to understand the limitations presented in Blimp and emulator when designing an OSD using Blimp's components.

The OSD hardware is capable of overlaying three type of drawing units on the top of the incoming video data: filled boxes (Fbox), text boxes (Tbox), and icon boxes (Ibox).

Fbox, Tbox and Ibox hardware limitation to consider: The OSD hardware can display per single video line:

- 10 Fbox
- 10 Ibox
- 10 Tbox

The OSD hardware is capable of displaying up to 10 Fboxes, 10 Iboxes, and 10 Tboxes on each line of video data.

There are also many levels of overlaying OSD data. It is possible to assign a different priority level to each OSD component, thus allowing the overlap of text on filled boxes or even of filled boxes on top of filled boxes. This can be used, for example, for menu overlay.

Filled boxes can be used to surround text boxes to give the structure of menus. There are a total of 64 possible filled boxes which can be displayed. Filled boxes are also configurable in terms of scale and color. Up to 8 different colors can be assigned to Fboxes and its border at any time. Filled boxes can be enabled and disabled to display or hide a given box.

Text boxes can contain up to 16 characters, with each character represented by a maximum of 16x16 pixels (including spaces). A total of 64 possible text boxes can be displayed and fully configurable. Text boxes can be enabled and scaled depending on the selected output format. Up to 16 different colors can be assigned to Tboxes at any time. The OSD supports up to 256 different characters which can be loaded into the internal RAM on power up. If more characters are required, it is possible to reload the character RAM using the high speed SPI interface. The software design had taken into account the capability to de-allocate characters from Font RAM when a string or characters from any component is disabled and no other component uses the character. High resolution font can be supported using Tboxes. While supporting, font characters occupy more character RAM as well as Tboxes depending on the font size.

Icon boxes are used to display the icon and images. There are 128 icons which can be displayed at a time and the ADV7625 can store 64 unique icons at any time. Each icon has a size of 8x8 pixels.

The ADV7625 can store 32 unique colors at any time for displaying the image.

1.1.5 OSD resolution scaling

The ADV7625 does not have an OSD scalar to fit any resolution according to input video. The OSD hardware allows for a general scaling value per Tbox, Fbox and Ibox of integer value from 1 to 15.

2 OSD API functions

This chapter provides information about pages in Blimp.

The following topics are covered:

2.1 Page API's

The pages implement events, can be hidden off or shown on the display as the user moves through the OSD, and it is also possible to have more than one page visible at the same time.

2.1.1 API's Show/Hide Focus Pages

This section shows the methods which can be used to control visibility and focus of the pages within Blimp. Table 3 is shown as Page APIs method.

Table 3 Page API's

API	Short Description
ADIAPI_OSDEgShowPage	Displays one page on screen
ADIAPI_OSDEgHidePage	Hides one page from being displayed on screen
ADIAPI_OSDEgSetFocusPage	Sets the focus to one page

2.1.1.1 2.1.1.1 ADIAPI_OSDEgShowPage

Syntax: ADIAPI_OSDEgShowPage (OsdFrameWnd* Page);

Displays one page on the screen.

Code window usage example:

```
OsdApi.ADIAPI_OSDEgShowPage(PageManager.InputSelection);
```

2.1.1.2 2.1.1.2 ADIAPI_OSDEgHidePage

Syntax: ADIAPI_OSDEgHidePage(OsdFrameWnd* Page);

Hides one page from being displayed on the screen.

Code window usage example:

```
OsdApi.ADIAPI_OSDEgHidePage(PageManager.Page1,0);
```

2.1.1.3 ADIAPI_OSDEgSetFocusPage

Syntax: ADIAPI_OSDEgSetFocusPage (OsdFrameWnd* Page);

Sets the focus to one page.

Code window usage example:

```
OsdApi.ADIAPI_OSDEgSetFocusPage(PageManager.InputSelection);
```

2.2 Other APIs

This section presents other APIs that can be used in Blimp pages and components.Table 4is shown as OSD APIs method.

Table 4Other OSD API's

API	Short Description
ADIAPI_OSDEgSetLanguage	Sets language in use within OSD
ADIAPI_OSDEgSetFocusComponent	Sets focus to a component

2.2.1 ADIAPI_OSDEgSetLanguage

Syntax: ADIAPI_OSDEgSetLanguage(OSD_LANGUAGES language);

Sets the language in use in the OSD.

Code window usage example:

```
OsdApi.ADIAPI_OSDEgSetLanguage(OSD_LANGUAGES.SPANISH);
```

2.2.2 ADIAPI_OSDEgSetFocusComponent

Syntax: ADIAPI_OSDEgSetFocusComponent (OSDControl name);

Sets the focus to one component. Note that only one component can receive the focus at a given time, and it remains set until it is set to another component.

Allow the user to move through an OSDListbox component called main Menu by setting focus to it

Code window usage example:

```
OsdApi.ADIAPI_OSDEgSetFocusComponent(mainMenu);
```

2.3 Pressed Key Event Flow Between Pages and Components

Some OSD components implement a *RemoteKeyPress* event: *OSDListbox* and *OSDProgressBar*. This event is triggered when the user presses a key. When this happens, the captured keystroke is sent to these event methods, where the user may define a custom response to the pressed key.

However, before the keystroke is sent to them, the page *RemoteKeyPress* event triggers first. If the user does not cancel here the received keystroke, and once the code within the event *Page_KeyPress* has executed, the specific event of each component then triggers.

Within the component *RemoteKeypress* event, something similar may happen. If the user does not cancel the received keystroke, once the code within the event has executed, the keystroke is passed to the component. However, this last step may or may not take place depending on the component and the pressed key. As mentioned later within the section of each component, some components (the ones which can take focus, that is, the components which implement a *RemoteKeypress* event), have some *hardcoded* keys. This means that the components will automatically 'react' to these keys, without the need for the user to manually define an action within the code. For example, an *OSDListbox* or *OSDMenubar* automatically allows the user to move through its items by just pressing the arrow keys. It is for these components and for these keys that the last step in the *RemoteKeypress→ Page → Component* event flow makes sense.

The pressed key flow for *OSDListbox* and *OSDProgressBar* components can be seen in Figure 2.

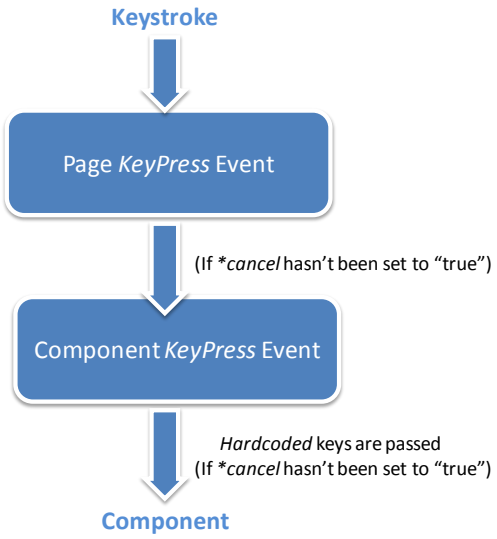


Figure 2. Pressed Key Event Flow

This keystroke flow between pages and components gives the OSD designer a lot of flexibility and resources to define the user-OSD interaction. For example, the *RemoteKeypress* event of a page could be used to capture certain keystrokes which, under certain circumstances, the user does not want to be received by any of the components in that page. Or, it could be used to perform some task before passing the received key to the *RemoteKeypress* event of the component. For component-specific processing of the keystrokes, the *RemoteKeypress* event of each component could then be used.

3 Using OSD Components

This chapter describes the OSD components, properties, methods and events.

The following topics are covered:

3.1 Common Properties

Every OSD component implements the common properties, which are listed in Table 5. Note that, although common to all the OSD components, these properties are only presented here and will not be included on each of the individual OSD components presented later.

Table 5Common Properties and Methods

Global Properties	Description
Name	Indicates name used to identify object
Visible	Gets or sets visibility status of component
LocationX	Gets or sets horizontal position of component
LocationY	Gets or sets vertical position of component
Priority	Sets overlay priority of component
Size	Sets size of component

Note that *Size* can only be assigned through the *Property Navigator* of the Blimp GUI, that is, it cannot be accessed or modified through the user code. This means it does not have an equivalent on the firmware. In other words, once the size of the components is set, they are fixed and cannot be modified in real time.

In the current version of the tool, ten different overlay priorities can be selected.

Figure 3 shows an example of the use of *Priority* property within a complex OSD design.

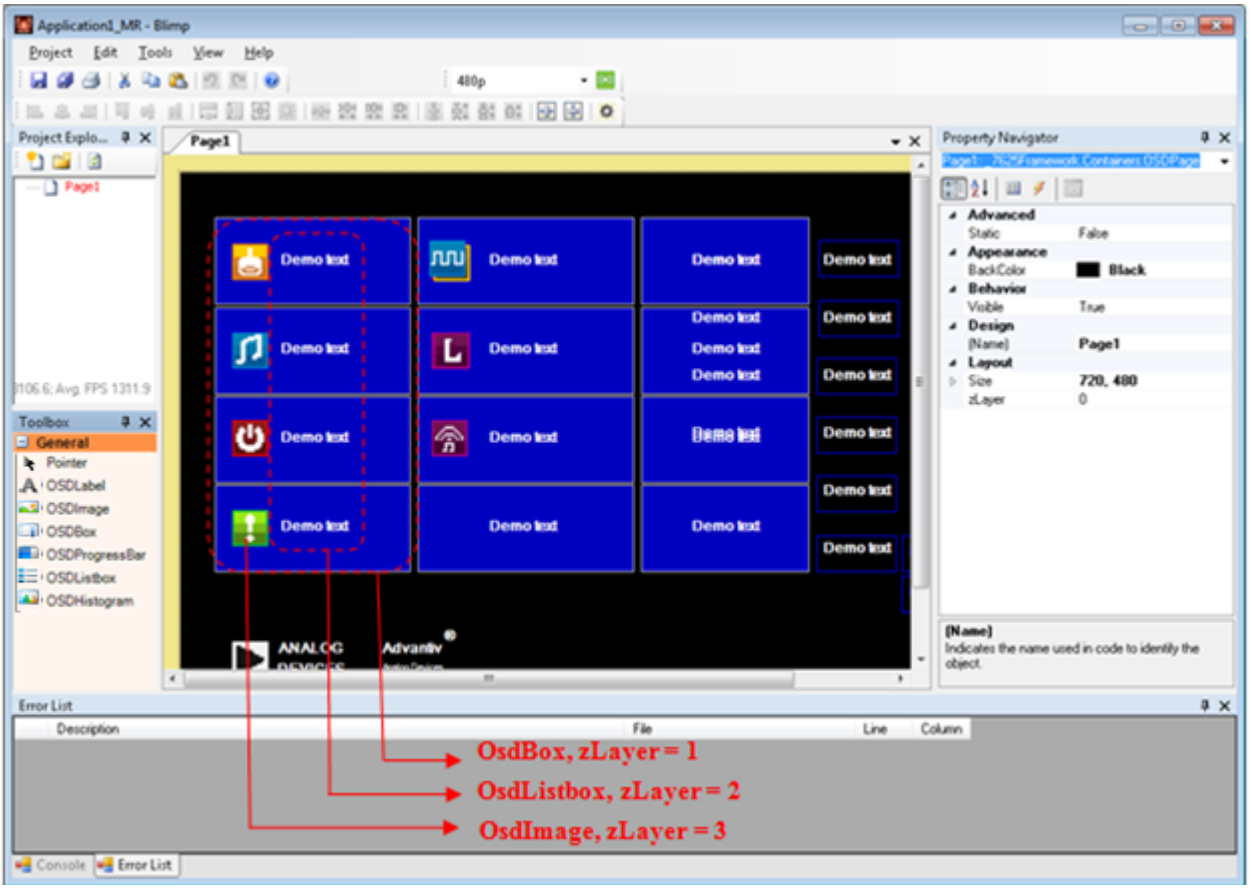


Figure 3: Example Usage of Priority Property (ADV7625shown)

Each property is now described in detail, indicating how each property is declared within the Blimp tool. An example is given for how to use the property in the code window.

3.1.1 Name

Description: Indicates the name used in the code to identify the object.

Property declaration:

```
public string Name {set;}
```

Auto complete: No

3.1.2 Visible

Description: Sets if the component is visible or hidden by default.

Selection: True or false

Auto complete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public bool Visible {get; set;}
```

Code window usage example:

```
osdListbox2.Visible = true;
```

3.1.3 Location(X,Y)

Description: Displays and sets the coordinates of the upper-left corner of the component relative to the upper-left coordinate container for the selected resolution. Note: Each resolution will have its own set of coordinates and size.

Range: UINT16

Auto complete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public short LocationX {get;}

public short LocationY {get;}
```

Code window usage example:

```
int locX, locY;

locX = osdListbox2.LocationX;

locY = osdListbox2.LocationY;
```

For Multi resolution, the location is stored in binary. So the location should not be changed at runtime.

3.1.4 Priority

Description: Sets the layer overlay priority of the component. The *priority* of the component shall fill the three LSBs of the 5-bit priority of Ibox, Tbox or Fbox.

Range: 0 – 7

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public UINT8 Priority {get; set;}
```

Code window usage example:

```
osdListbox2.Priority= 1;
```

Z layer priority has some limitation for the combination of same components. For overlapping two fboxes or tboxes or iboxes, the lower z layer value overlaps with a higher number, means smaller number will be drawn on top of bigger number. If there are two fboxes or tboxes or iboxes with the same priority number, the one with the lowest address at the instruction RAM will be drawn on top.

3.1.5 Size (Width, Height)

Description: Displays and sets the size of the control in pixels for the selected resolution. Each resolution will have their own set of coordinates and size. Please note that it cannot be changed in code window.

Range: UINT16

Autocomplete: No

Property declaration:

```
public shortSize {get; set;}
```

For Multi resolution, the size is stored in binary. So the size should not be changed at runtime.

3.2 Enumeration Types

These types are accessed by some properties within the OSD components or pages.

Table 6Enumeration Types

Enumeration	Description
OSDContentAlignment	Defines how different components are horizontal and vertically aligned
OSDAnimationType	Defines behavior of OSDImage animation
OSDOrientation	Orientates component vertically or horizontally
Direction	Defines the direction of an OSDProgressBar component
OSDListboxMode	Defines behavior of selected items within an OSDlistbox component

3.2.1 OSDContentAlignment

Description: Defines how the different components are horizontal and vertically aligned.

Property declaration:

```
public enum {
  TOPLEFT = 0,
  TOPCENTER,
  TOPRIGHT,
  MIDDLELEFT,
  MIDDLECENTER,
  MIDDLERIGHT,
  BOTTOMLEFT,
  BOTTOMCENTER,
  BOTTOMRIGHT
```

```
} OSD_CONTENTALIGNMENT;
```

TopLeft: Content is vertically aligned at the top, and horizontally aligned on the left.
TopCenter: Content is vertically aligned at the top, and horizontally aligned at the center.
TopRight: Content is vertically aligned at the top, and horizontally aligned on the right.
MiddleLeft: Content is vertically aligned in the middle, and horizontally aligned on the left.
MiddleCenter: Content is vertically aligned in the middle, and horizontally aligned at the center.
MiddleRight: Contentis vertically aligned in the middle, and horizontally aligned on the right.
BottomLeft: Content is vertically aligned at the bottom, and horizontally aligned on the left.
BottomCenter: Content is vertically aligned at the bottom, and horizontally aligned at the center.
BottomRighT: Content is vertically aligned at the bottom, and horizontally aligned on the right.

3.2.2 OSDAnimationType

Description: Defines the behavior of an *OSDImage* animation.

Property declaration:

```
public enum {
 ONETIME = 0,
  LOOP,
  BOUNCE,
} OSD_ANIMATIONTYPE;
```

ONETIME: Animation will only be executed once.

LOOP: Animation will be executed continuously.

BOUNCE: Animation will be executed only once from the first tile until the last tile, then from last tile back to the first tile.

3.2.3 OSDOrientation

Description: Orientates vertically or horizontally the component. Can be used on the *OSDHistogram*component. This is usually set through the GUI although it may also be changed at run time.

Property declaration:

```
public enum {
  VERTICAL = 0,
  HORIZONTAL
} OSD_ORIENTATION;
```

Vertical:Vertical orientation

Horizontal:Horizontal orientation

For example, assuming that osdMainMenu is a vertically-orientated OSDHistogram component, its orientation can be changed to horizontally-orientated during runtime through

Code window usage example:

```
osdMainMenu.Orientation = OSD_ORIENTATION.Horizontal;
```

3.2.4 Direction

Description: Defines the directionof an*OSDProgressBar component*.

Property declaration:

```
typedefenum

{

LeftToRight,

RightToLeft,

 TopToBottom,

 BottomToTop

} Direction;
```

LeftToRight: Left to right direction

RightToLeft: Right to left direction

TopToBottom: Top to bottom direction

BottomToTop: Bottom to right direction

3.2.5 OSDListboxMode

Description: Used to define the behavior of the selected items within an *OSDlistbox* component. This is usually set through the GUI although it may also be changed at run time.

Property declaration:

```
public enum {
  SCROLLING_SINGLE_SELECTION_ITEM_FIRST,

    SCROLLING_SINGLE_SELECTION,

} OSD_LISTBOXMODE;
```

SCROLLING_SINGLE_SELECTION: Only one selected item per listbox.

SCROLLING_NON_SELECTION: Noselected item per listbox.

Code window usage example:

```
public void Load()
{
   VSPListbox.TotalItems = 3;
   VSPListbox.VisibleItems = 3;

   VSPListbox.ItemText[0] = "MNR";
   VSPListbox.ItemText[1] = "BNR";
   VSPListbox.ItemText[2] = "Sharpness Filter";

   OsdApi.ADIAPI_OSDEgSetFocusComponent(VSPListbox);
}
```

The Blimp current version only supports `SCROLLING_SINGLE_SELECTION` and `SCROLLING NON SELECTION` mode.

3.2.6 Language Enumeration Types

Description: This is a special form of enumeration type, which is created dynamically by Blimp when in a Multilanguage OSD. It consists of an *enum* with as many languages as defined by the user. For example, Blimp could automatically generate the following enumeration when compiling an OSD. It is stored in the *blimp_language.h* file, within the *Release* folder, and should not be modified by the user.

```
typedef enum _enumLanguageEnum
{
  ENGLISH,
SPANISH,
  JAPANESE,
  DEUTSCH,
  ITALIAN,
} LanguageEnumeration;
```

3.3 Pages

Pages are the containers for the OSD components. Pages are created on Blimp by using the *Project Explorer,* which is also used for accessing the code window within each page.

The pages implement events, which can be hidden or shown on the display as the user moves through the OSD. It is also possible to have more than one page visible at the same time.

Pages need to have focus set on them in order to give focus to any component within it. For example, if the user switches from one page to other and sets the focus to an *OSDListbox* contained on the second page, the *OSDListbox* will not be able to receive any key input from the user unless the focus has also been set to the current page.

By default, visibility and focus are automatically given to the main page of the OSD, which is the first page created on Blimp. This can be changed by changing the start-up page as defined in Blimp User Manual.

3.3.1 Page Events

Pages also implements events, in the same manner as OSD components do.

When the starting page loads, the events :*Activate*, *Load* and *VisibleChanged* will get triggered by default. This is the expected behavior since the page has been loaded, made visible and set the focus on automatically by Blimp.

Table 7Pages Events

Page Event	Short Description
OnActivate	Occurs when page gains main active focus
OnDeactivate	Occurs when page loses main active focus
Load	Occurs when page has finished loading
RemoteKeyPress	Occurs when page has focus and user presses and releases a key
OnEnable	Occurs when page visibility changes to TRUE
OnDisable	Occurs when page visibility changes to FALSE

3.3.1.1 OnActivate()

Description: Occurs when the page gains the main active focus.

Code window usage example:

```
private void Page1_OnActivate()
{
    osdLabel1.Text = "Activate page event triggered";
}
```

3.3.1.2 OnDeactivate()

Description: Occurs when the page loses the main active focus.

Code window usage example:

```
private void Page1_OnDeactivate()
{
    osdLabel1.Text = "De-activate page event triggered";
}
```

3.3.1.3 Load()

Description: Occurs when the page has finished loading.

Code window usage example:

```
private void Page1_Load()
{
    osdLabel1.Text = "Load page event triggered";
}
```

3.3.1.4 RemoteKeyPress

Syntax: RemoteKeyPress (Byte *keyCode, Boolean *cancel)

Description: Occurs when the page has focus and the user presses and releases a key. See the Pressed Key Event Flow Between Pages and Components section for more information.

Parameters:

***keyCode:**Pointer to the variable which stores the pressed key. User can modify its value to manually assign a different keycode to the keystroke.

***cancel**:Pointer to the variable which determines if the pressed key is further processed or not. If set to “false”, the KeyPress event (if any) triggers, and the component does not receive the keystroke (for example, an OSDListbox does not receive a “down” key).

Code window usage example:

```
private void Page1_RemoteKeyPress(Byte* keyCode, Boolean *cancel)
{
    if (*keyCode == 38) //Up key
    {
        OsdApi.ADIAPI_OSDEgShowPage(PageManager.Page2);
        OsdApi.ADIAPI_OSDEgHidePage(PageManager.Page1);
       OsdApi.ADIAPI_OSDEgSetFocusPage(PageManager.Page2);
    }
}
```

3.3.1.5 OnEnable()

Description: Occurs when the page visibility changes to TRUE.

Code window usage example:

```
private void Page1_OnEnable()
{
    osdLabel1.Text = "Visibility of the page event triggered";
}
```

3.3.1.6 OnDisable()

Description: Occurs when the page visibility changes to FALSE.

Code window usage example:

```
private void Page1_OnDisable()
{
    osdLabel1.Text = "Visibility of the page event triggered";
}
```

3.4 OSDLabel

OSDLabel component is used to display text using Tbox.

3.4.1 OSDLabel Properties

Table 8OSDLabel Properties

Property or Method	Short Description
Text	Sets the default text
FgColor	Sets the foreground color of the text.
TextAlign	Sets the alignment of the text.
Font	Sets the font.
ConstText	Sets a string from the multilanguage string table. The StringID can be used to specify the string using StringManager.
Orientation	Sets the direction of the text.
MaxLength	Sets the maximum number of characters the Text property can contain.
LineSpacing	Sets Line spacing for multiline mode in pixels from bottom to bottom of each line
MultilineEnabled	Sets if component can display more than one line of text
setTextFormat()	Sets a formatted text string in ascii. This method is very similar to the sprint function in C.
TextW	Sets a unicode string from a buffer of type uint16[].
readTextW()	Gets the text of the OSDLabel component and stores it into a buffer of unicode fomat uint16[].

Scrolling • **ScrollingEnabled**	Sets if text scrolling is enabled
• **ScrollingSpeed**	Sets speed of scrolling
• **ScrollingIntervalTime**	Sets the amount of time the control will wait until scrolling is restarted.(in milliseconds)
• **Spacing**	Sets spacing between text during scrolling
• **LeftToRight**	Sets the direction of scrolling.
• **ScrollingTextRepeat**	Causes the text to concatenate after the scrolling spacing
• **ScrollingBounceBack**	Causes the direction of the scrolling text to alternate direction when text reaches the end of the display area
• **ScrollingOffDisplay**	Text will start scrolling from outside of display area and end scrolling

3.4.1.1 FgColor

Description: Sets the foreground color of the text.

Range: See RGB color selection entry limit

Auto complete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public UINT32 Color {get; set;}
```

To set the 32 bits value {A,R,G,B}

Code window usage example:

```
Sets the color to full opaque black
osdLabel1.Color = 0xFFFFFFFFu;
```

3.4.1.2 Font

Description: Sets the font. It is possible to assign new fonts in runtime through the code window. Note that the selected font needs to be installed on the machine running Blimp, although there are a couple of font formats that are not allowed in Blimp even though the operating system fully supports them:

- Device-specific/printer fonts
- Raster/bitmap fonts
- Open Type (erratic; Microsoft fonts are allowed, Adobe fonts are not allowed)
- Type 1 fonts

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public FontFont {get; set;}
```

FontStyle: Regular, Bold, Italic, Underlined, Strikeout. You can as well assign a current font to a new control

Code window usage example:

```
osdLabel1.Font = new Font("Arial", 15f, FontStyle.Regular);
osdLabel1.Font = osdLabel2.Font;
```

3.4.1.3 TextAlign

Description: Sets the alignment of the text.

You can use this property to align the text within an *OSDLabel* to match the layout of controls on your page. This property is usually set in the canvas window through the *Property Navigator*, although it can also be used in the scripting window. For example, if your controls are located to the right edge of the labels, you can set the *TextAlign* property to one of the right-aligned horizontal alignments (TOPRIGHT, MIDDLERIGHT, and BOTTOMRIGHT) and the text will be aligned with the right edge of the labels to align with your controls.

Selection:

TopLeft	TopCenter	TopRight
MiddleLeft	MiddleCenter	MiddleRight
BottomLeft	BottomCenter	BottomRight

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public OSD_CONTENTALIGNMENT TextAlign {get; set;}
```

To align the text to the top right corner

Code window usage example:

```
osdLabel1.TextAlign = OSD_CONTENTALIGNMENT.MiddleCenter;
```

3.4.1.4 MaxLength

Description: Sets the maximum amount of Unicode characters the *OSDLabel* can contain at any time. It is usually set in the GUI although it may also be changed during run time.

Range: UINT32

Auto complete: No

3.4.1.5 Text

Description: Sets the default text. Note that this property is limited to ASCII characters. If string length is greater than MaxLength, the string shall be truncated to MaxLength characters. When OSD running and no scrolling is selected, if string length is greater than component display area size, the Label shall append "…" at the end of the OSDLabel display region and truncate characters as needed.

Autocomplete: yes, Gets/Sets, Sets an ascii string from a buffer of type char[].

Property declaration:

```
public string Text {set;}
```

Property of two OSDLabel when the OSD state gets changed.

Code window usage example:

```
descriptionLabel.Text = "Select the input connector.";
titleLabel.Text = "INPUT SELECTOR";
```

3.4.1.6 ConstText

Description: Sets a string from the Multilanguage string table. The StringID can be used to specify the string using StringManager. The definition of the string has to be entered through the *Language Settings* menu in Blimp. For more information, refer to the Language configuration section of the Blimp User Manual. Section 3.11

Method declaration:

```
public UINT32 ConstText{set;}
```

Where INPUT_SELECTION is the defined stringID which represents the appropriate text string which will appear into the component when the desired language is chosen

Code window usage example:

```
osdLabel1.ConstText = StringManager.INPUT_SELECTION;
```

Note: stringID will always be uppercase even if defined as lowercase.

3.4.1.7 Orientation

Description This property is used to sets direction of the text within an *OSDLabel* to match the layout of controls on your page. The direction of the text can be either horizontal or vertical as per the selection in *Property Window*.

Property declaration:

```
public OSD_ORIENTATIONOrientation {get; set;}
```

Auto complete: No

3.4.1.8 MultilineEnabled

Gets or sets if the component can display multiple lines of text. Note that enabling Multiline disables the horizontal scrolling. When setting this property to true, it is also possible to use new line escape character, "\n", to manually create a new line of text. If the box height that defines the OSDLabel component is big enough to fit the lines of text, there is no vertical scrolling. If not, vertical scrolling occurs.

Method declaration:

```
public bool MultilineEnabled{get; set;}
```

Code window usage example:

```
osdLabel1.MultilineEnabled = true;
```

Note: Only top left alignment is currently supported for multi-line feature.

3.4.1.9 LineSpacing

Sets Line spacing for multiline mode in pixels from bottom to bottom of each line

Autocomplete: No

Method declaration:

```
public UINT8 LineSpacing{set;}
```

3.4.1.10 setTextFormat

Sets a formatted text string in ascii. This method is very similar to the sprint function in C. It is used to deal with the string concatenation and representation which needs to be done in an ANSI-C compatible fashion.

Method declaration:

```
public INT32 setTextFormat(string format, params Object[] args);
```

Parameters:

format: String to be formatted and stored. Ordinary ASCII characters excluding % are directly converted to the output string. Each conversion command will fetch one parameter in the order the commands are added to the format string.

Since the string will be stored in the stack, in order to avoid stack overflowing, there is a limitation to the number of characters that can be used in the string. By default, the firmware limits it to 512 bytes through MAX_TEXTFORMAT_LENGTH define directive. This value is a good reference value, big enough for most controls while small enough to not collapse the stack.

args: List of the data required for the conversion commands.

Table 9: List of supported formats in setTextFormat

%c	a character with the given number
%s	a string
%d	a signed integer, in decimal
%u	an unsigned integer, in decimal
%o	an unsigned integer, in octal
%x	an unsigned integer, in hexadecimal
%e	a floating-point number, in scientific notation
%f	a floating-point number, in fixed decimal notation
%g	a floating-point number, in %e or %f notation

Code window usage examples:

```
osdLabel.setTextFormat("%s has been concatenated to
%d", "FIRST", "SECOND");
```

3.4.1.11 TextW

Sets a unicode string from a buffer of type uint16[].

Property declaration:

```
public string TextW {set;}
```

Code window usage example:

```
private void ReadBufferString()
{
    ushort[] buffer = new ushort[15];

    osdLabel1.Text = "Hello World";
    osdLabel2.Text = "Unitialized";

 //Read 11 characters from osdLabel1 and put the read string into
buffer
    osdLabel1.readTextW(buffer,11);

    //osdLabel2 reads now "Hello World".
    osdLabel2.TextW = buffer;

}
```

3.4.1.12 readTextW

Gets the text of the OSDLabel component and stores it into a buffer of unicode format uint16.

Method declaration:

```
public void readTextW(ushort* buffer, ushort size);
```

Parameters:

buffer:Array where the read characters are going to be stored. It should be long enough to hold the value set in the *size* parameter.

size:Space which needs to be reserved in the buffer for the string being read. It could also be seen as the numbers of characters (plus a null termination) that are going to be read from the label. For example, if the label length is 11, size should be set to 12.

Code window usage example:

```
private void ReadLabel()
{
    //The buffer needs to be ushort in order to hold a Unicode string
    ushort[] buffer = new ushort[15];

    osdLabel1.Text = "Hello World";
    osdLabel2.Text = "Unitialized";

    //Read 11 characters from osdLabel1 and put the read string into
buffer
    osdLabel1.readTextW(buffer,12);

    //osdLabel2 reads now "Hello World".
    osdLabel2.TextW = buffer;
}
```

3.4.2 OSDLabel Scrolling Property

3.4.2.1 ScrollingEnabled

Description: Sets if text scrolling is enabled. When scrolling is enabled, the Label text is repeated continuously with a space set by *ScrollingSpacing* in between. Setting *ScrollingEnabled* to '0' disables completely any scrolling method declaration:

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
public boolScrollingEnabled{get; set;}
```

Code window usage example:

```
osdLabel1.ScrollingEnabled = TRUE;
```

3.4.2.2 LeftToRight

Description: Sets the direction of the scrolling, either *LeftToRight* or *RightToLeft*.If set to true, the text scrolls from left to right; if set to false, scrolls from right to left.

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
public bool LeftToRight{ get; set;}
```

Code window usage example:

```
osdLabel1.LeftToRight = TRUE;
```

3.4.2.3 ScrollingSpeed

Description: Sets the speed of the scrolling. The speed is defined in pixels per second

Range: 0 – 255

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
public UINT8ScrollingSpeed{ get; set;}
```

Code window usage example:

```
osdLabel1.ScrollingSpeed = 20;
```

3.4.2.4 ScrollingIntervalTime

Description: Sets the amount of time (in milliseconds) the control waits until scrolling is restarted.

Autocomplete: No

Method declaration:

```
public UINT16 ScrollingIntervalTime{get; set;}
```

3.4.2.5 ScrollingSpacing

Description: Gets or sets the spacing between text when text is repeated in scrolling in pixels.

Method declaration:

```
public UINT16 ScrollingSpacing{get; set;}
```

Auto Complete: No

3.4.2.6 ScrollingBounceBack

Description: Causes the direction of the scrolling text to alternate direction when text reaches the end of the display area.

Auto Complete: No

Method declaration:

```
public bool ScrollingBounceBack {get; set;}
```

3.4.2.7 ScrollingTextRepeat

Description: Causes the text to concatenate after the scrolling spacing

AutoComplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
public bool ScrollingTextRepeat {get; set;}
```

Code window usage example:

```
osdLabel1.TextRepeat = TRUE;
```

3.4.2.8 ScrollingOffDisplay

Description: Text will start scrolling from outside of display area and end scrolling when text is completely outside display area

Auto Complete: No

Method declaration:

```
public bool ScrollingOffDisplay{get; set;}
```

3.4.3 OSDLabel Events

Table 10OSDLabel Events

Event	Short Description
ScrollingFinish	Triggerred when text scrolling has finished.

3.4.3.1 ScrollingFinish

Description: This event Triggered when text scrolling has finished. Note that the event triggers just in the moment the text stops scrolling, that is, before waiting for the period defined by *ScrollingIntervalTime.*

Auto complete: No

3.5 OSDListbox

The *OSDListbox* displays a list of different text strings through which the user can move and select. None, one, or more items may be selected depending on the *Mode* property. If the number of items exceeds the maximum number of items visible at a time, the *OSDListbox* develops scrolling functionality. The color of the text for this component can change depending on the state of the item (default, highlighted, selected and disabled). This can be set from the properties *DefaultTextColor*, *HighlightedTextColor*, *SelectedTextColor and DisabledTextColor*.

This component may be the target of the page focus, which needs to be set prior to receiving any keystroke input from the user. In addition, it has four events associated with it.

The keys used to move through the *OSDListbox* are described in Table 11.

Table 11Hardcoded Keys in OSDListbox Component

Key	Key Code (decimal)	Function
Up Arrow	38	Move up the list
Down Arrow	40	Move down the list
Spacebar	32	Select an item
Enter	13	Select an item

3.5.1 OSDListbox Properties

Table 12OSDListbox Properties

Property or Method	Short Description
TotalItems	Sets the maximum number of items that the list can contain.
VisibleItems	Sets the number of items that list will display.
Font	Sets the font.
TextAlign	Sets the alignment of the text.
Color • **DefaultTextColor** • **HighlightedTextColor** • **SelectedTextColor** • **DisabledTextColor**	 Sets the default color for the list items text Sets for the color for the highlighted item in the list Sets for the color for the selected item(s) in the list Sets for the color for disabled items in the list.
ItemText	Sets an item text as a null terminated ascii string from a buffer of type char[]
ItemConstText	Sets an text as a string from the multilanguage string table. The StringID can be used to specify the string using StringManager.
Mode	Sets scrolling and selection behavior of listbox
ItemHeight	Sets the height of each item in the list box in pixels.
VerScrollingSpeed	Sets speed of vertical scrolling animation
MaxLength	Sets the maximum number of characters the Text property can contain.
FocuIndex	Gets or sets the current highlighted item index
SelectedIndex	Gets the current selected item index
HighlightedSlot	Gets offset of *OSDListbox.*It does nottake hidden items into account.
RollBack	Gets or sets whether the list will keep scrolling through the first item on the list once the last one has been reached and vice versa.
setItemTextFormat()	Sets an item text as a formatted text string in ascii. This method is very similar to the sprint function in C.
DisabledItem	Sets items that appear disabled in the list.
SkipDisabledItems	If True, selection will jump over disabled items when scrolling through listbox items.
HiddenItems	Sets hide property of an item.

Property or Method	Short Description
Scrolling • **ScrollingEnabled** • **ScrollingSpeed** • **ScrollingIntervalTime** • **ScrollingSpacing** • **LeftToRight** • **ScrollingBounceBack**	When scrolling is enabled, the Label text shall be repeated continuously with a space set by ScrollingSpacing in between. Sets the speed of the horizontal scrolling animation (pixels/frame). 0 to disable. Sets the amount of time the control will wait until scrolling is restarted. Sets the spacing between the text during scrolling mode. (pixels) Sets the direction of the horizontal scrolling text. Causes the direction of the scrolling text to alternate direction when text reaches the end of the display area

3.5.1.1 TotalItems

Description: Sets maximum number of items that the list can contain.

Note that *TotalItems* must always be bigger than or equal to the *VisibleItems* property

Property declaration:

```
public byte TotalItems{get; set;}
```

Code window usage example:

```
osdListbox1.TotalItems = 4;
```

3.5.1.2 VisibleItems

Description: Sets the number of items that the list will display. If VisibleItems <TotalItems, the listbox can be scrolled.

Property declaration:

```
public byte VisibleItems{get; set;}
```

Code window usage example:

```
osdListbox1.VisibleItems = 2;
```

3.5.1.3 Font

Description: Sets the font. It is possible to assign new fonts at runtime through the code window. Note that the selected font needs to be installed on the machine running Blimp, although there are a couple of font formats which are not allowed in Blimp, even though the operating system fully supports them:

Device-specific/printer fonts
Raster/bitmap fonts
Open Type (erratic; Microsoft fonts are allowed, Adobe fonts are not allowed)
Type 1 fonts

Property declaration:

```
public Font Font{set;}
```

Code window usage example:

```
osdListbox1.Font = new Font("Tahoma", 15f, FontStyle.Regular);
```

3.5.1.4 TextAlign

Description: Sets the alignment of the text. All the items within the list align in the same way. This property is usually set in the canvas window through the *Property Navigator*, although it can also be used in the scripting window.

Property declaration:

```
public OSD_CONTENTALIGNMENT TextAlign {get; set;}
```

Align all the items within the OSDListbox to the middle left

Code window usage example:

```
osdListbox1.TextAlign = OSD_CONTENTALIGNMENT.MiddleLeft;
```

3.5.1.5 Color

DefaultTextColor

Description: Sets the default color for the list items text. This is usually set through the GUI although it may be changed at run time.

Range: RGB color selection. RGB (red, green, and blue) refers to a system for representing the colors to be used on a computer display. Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum. Each level is represented by the range of decimal numbers from 0 to 255 (256 levels for each color),

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public UINT32 DefaultTextColor{get; set;}
```

Set default text to Green color

Code window usage example:

```
osdListbox1.DefaultTextColor = 0x00FF00u;
```

HighlightedTextColor

Description: Sets for the color for the highlighted item in the list. This is usually set through the GUI although it may be changed at run time.

Property declaration:

```
public UINT32 HighlightedTextColor{get; set;}
```

Code window usage example:

```
osdListbox1.HighlightedTextColor = 0xFFFFFF00u;
```

SelectedTextColor

Description: Sets the color for the selected item(s) in the list. This is usually set through the GUI although it may be changed at run time.

Property declaration:

```
public UINT32 SelectedTextColor{get; set;}
```

Code window usage example:

```
osdListbox1.SelectedTextColor = 0xFFFFFFFFu;
```

DisabledTextColor

Description: Gets or sets for the color for disabled items in the list.

Property declaration:

```
public UINT32 DisabledTextColor{get; set;}
```

Code window usage example:

```
   //Set disabled item color to gray
osdListbox1.DisabledTextColor = 0xFFAAAAAAu;
```

3.5.1.6 ItemText

Description: Sets an item text as a null terminated ascii string from a buffer of type char[]. Note that this property is limited to ASCII characters.

Property declaration:

```
public string[] ItemText{get;}
```

Code window usage example:

```
osdListbox1.ItemText[0] = "First item of the list";
```

3.5.1.7 ItemConstText

Description: Sets text as a string from the Multilanguage string table. The StringID can be used to specify the string using StringManager. The definition of the string has to be entered through the *Language Settings* menu in Blimp. Refer Blimp user manual section 3.11 for more information.

Method declaration:

```
public UINT32ItemConstText{get;}
```

Where AUDIO_SETTINGS is the defined stringID which represents the appropriate text string which will appear into the item text when the desired language is chosen

Code window usage example:

```
osdLabel1.ItemConstText[1] = StringManager.AUDIO_SETTINGS;
```

StringID will always be uppercase even if defined as lowercase.

3.5.1.8 setItemTextFormat

Description: Sets an item text as a formatted text string in ascii. This method is very similar to the *sprintf* function in C. It is used to deal with the string concatenation and representation which needs to be done in an ANSI-C compatible fashion. Refer Table 9 for list of supported formats.

Method declaration:

```
UINT32 setItemTextFormat(UINT8 index, string format, params Object[]
args);
```

Paramenters:

index: Index of the item within the list.

format: String to be formatted and stored. Ordinary ASCII characters excluding % are directly converted to the output string. Each conversion command will fetch one parameter in the order the commands are added to the format string.

Note: Since the string will be stored in the stack, in order to avoid stack overflowing, there is a limitation to the number of characters which can be used in the string. By default, the firmware limits it to 512 bytes through MAX_TEXTFORMAT_LENGTH define directive. This value is a good reference value, big enough for most controls while small enough to not collapse the stack.

args: List of the data required for the conversion commands.

Code window usage example:

```
//It can useful, for example, for initializing long osdListboxes

byte i = 0;

    public void Load()
    {
        for(i=0;i<3;i++)
        {
            osdListbox1.setItemTextFormat(i,"HDMI Input %d",i);
        }
   OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
    }

//Also, we could use left and right arrow keys to create a kind of
submenu effect
    byte i = 0;

    public void Load()
    {
        osdListbox1.ItemText[0] = "Select HDMI Input";
        OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
    }
    private void osdListbox1_RemoteKeyPress(Byte *keyCode, Boolean
*cancel)
    {
        if (osdListbox1.FocusIndex == 0)
        {
            if (*keyCode == 39) //Right Arrow
            {
                i++;
                if (i>3)
                    i = 0;
                osdListbox1.setItemTextFormat(0,"HDMI Input %d",i+1);
            }
```

```
else if (*keyCode == 37) //Left Arrow
            {
                i--;
                if (i==255)
                    i = 3;
                osdListbox1.setItemTextFormat(0,"HDMI Input %d",i+1);
            }
    }
```

3.5.1.9 Mode

Description: Sets the scrolling and selection behavior of the listbox.

Property declaration:

```
public OSD_LISTBOXMODE Mode{get; set;}
```

Code window usage example:

```
osdListbox1.Mode = OSD_LISTBOXMODE.SCROLLING_SINGLE_SELECTION;
```

`SCROLLING_SINGLE_SELECTION` mode and `SCROLLING_NON_SELECTION` only is supported at the current version of the tool.

3.5.1.10 ItemHeight

Description: Sets the height of each item in the list box in pixels. The bigger the item size, the bigger the font size which could be used on the component. This is usually set through the GUI although it may be changed on run time.

Units: 8-bit integer

Property declaration:

```
public byteItemHeight{get; set;}
```

Code window usage example:

```
osdListbox1.ItemHeight = 30;
```

3.5.1.11 VerScrollingSpeed

Description: Sets the speed of the vertical scrolling animation through items. A value of '0' means vertical scrolling disabled.

Range[50 - 5000 ms]

Defaults shall be medium and option list as below can be selected:
Very fast - 250
Fast - 500
Medium - 1000
Slow - 1500
Very slow - 2000
A custom value (50 – 5000) can also be set directly.

Auto-complete: Yes

Method declaration:

```
public UINT32 VerScrollingSpeed{get; set;}
```

Code window usage example:

```
osdListbox1.VerScrollingSpeed = 20;
```

3.5.1.12 MaxLength

Description: Sets the maximum number of characters the Text property can contain. This property is usually set in the GUI although it may also be changed during run time.

Auto complete: No

Method declaration:

```
public UINT32 MaxLength{get; set;}
```

3.5.1.13 FocusIndex

Description: Gets or sets the current highlighted item index

Property declaration:

```
public UINT8 FocusIndex{get; set;}
```

Code window usage example:

```
osdListbox1.FocusIndex = 2;
```

3.5.1.14 Selectedndex

Description: Gets the current selected item index

Property declaration:

```
public UINT8 SelectedIndex {get; set;}
```

Code window usage example:

```
int index;
index = osdListbox1.SelectedIndex;
```

3.5.1.15 RollBack

Description: Gets or sets whether the list will keep scrolling through the first item on the list once the last one has been reached and vice versa.

Property declaration:

```
public bool Rollback{get;set;}
```

Code window usage example:

```
osdListbox1.RollBack = TRUE;
```

3.5.1.16 DisabledItems

Description: Gets or sets items that appear disabled in the list. If *DisabledItem* property is set, an item within the list will not be selectable, and the cursor will jump across to the next item on the list if *SkipDisabledItems* is set to true.

Property declaration:

```
public bool[] DisabledItems{get; set;}
```

Code window usage example:

```
osdListbox1.DisabledItems[1] = false;
```

3.5.1.17 SkipDisabledItems

Description: If true, selection will jump over disabled items when scrolling through listbox items. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
public boolSkipDisabledItems{get; set;}
```

3.5.1.18 HiddenItems

Description: Sets or Gets items that will not appear in the list. Sometimes the OSD designer may not want to disclose to the user some of the options on the menu (depending, for example, on other previous choice). This can be done through the use of the *HiddenItems* property.

Property declaration:

```
public bool[] HiddenItems{get; set;}
```

Code window usage example:

```
//In the following listbox, item 1 will not be draw thus being
invisible. The user will not be able to //select it, as it will always
navigate from item 0 to item 2 and from item 2 to items 0
osdListbox.TotalItems = 3;
osdListbox.VisibleItems = 3;
osdListbox.HiddenItems[1] = true;
```

3.5.1.19 HighlightedSlot

Description: Gets the offset of the *OSDListbox*items. It reads the offset between the visible first element of the list and the currently highlighted one. This offset does not take into account hidden items (see *OffsetIndex* property for more information). If the list does not implement any hidden items, *HighLightedSlot* reads the same as *OffsetIndex* property.

Property declaration:

```
//Create a list with scrolling and one hidden item

public UINT8 HighlightedSlot{get;}
```

Code window usage example:

```
unsafe public partial class Page1 : _7625Emulator.IPage
{
    public void Load()
    {
    }
    public void Dispose()
    {
```

```
    }

    private void Page1_Load()
    {
        osdListbox1.TotalItems = 7;
        osdListbox1.VisibleItems = 4;

        osdListbox1.ItemText[0] = "1st Item";
        osdListbox1.ItemText[1] = "2nd Item";
        osdListbox1.ItemText[2] = "3rd Item";
        osdListbox1.ItemText[3] = "4th Item";
        osdListbox1.ItemText[4] = "5th Item";
        osdListbox1.ItemText[5] = "6th Item";
        osdListbox1.ItemText[6] = "7th Item";

        osdListbox1.HiddenItems[2] = true;

        OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
    }

    private void osdListbox1_HighlightedItemChanged(Byte index)
    {
        osdLabel1.setTextFormat("Focus index
is %d",osdListbox1.FocusIndex);
        osdLabel2.setTextFormat("Offset index
is %d",osdListbox1.OffsetIndex);
        osdLabel3.setTextFormat("Highlighted slot
is %d",osdListbox1.HighlightedSlot);
    }
}
```

3.5.2 ScrollingProperty

3.5.2.1 ScrollingEnabled

Description: Sets if the scrolling functionality (horizontal) is enabled. When scrolling is enabled, the Label text is repeated continuously with a space set by *ScrollingSpacing* in between. Setting *ScrollingEnabled* to '0' disables the scrolling functionality completely.

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
public boolScrollingEnabled{get; set;}
```

Code window usage example:

```
osdListbox1.ScrollingEnabled = TRUE;
```

3.5.2.2 LeftToRight

Description: Sets the direction of the scrolling, either LeftToRight or RighttoLeft. If set to true, the text scrolls from left to right; if set to false, scrolls from right to left.

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
public bool LeftToRight{ get; set;}
```

Code window usage example:

```
osdListbox1.LeftToRight = TRUE;
```

3.5.2.3 ScrollingSpeed

Description: Sets the speed of the horizontal scrolling animation (pixels/frame). 0 to disable.

Range: 0 – 255

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Method declaration:

```
publicUINT16ScrollingSpeed{ get; set;}
```

Code window usage example:

```
osdListbox1.ScrollingSpeed = 20;
```

3.5.2.4 ScrollingIntervalTime

Description: Sets the amount of time (in milliseconds) the control waits until scrolling is restarted.

Autocomplete: No

Method declaration:

```
public boolScrollingIntervalTime{get; set;}
```

3.5.2.5 ScrollingSpacing

Description: Sets the spacing between the text during scrolling mode. (pixels)

Method declaration:

```
public UINT16ScrollingSpacing{get; set;}
```

Auto Complete: No

3.5.2.6 ScrollingBounceBack

Description: Causes the direction of the scrolling text to alternate direction when text reaches the end of the display area

Auto Complete: No

Method declaration:

```
public boolScrollingBounceBack {get; set;}
```

3.5.3 OSDListbox Events

Table 13OSDListbox Events

Event	Short Description
HighlightedItemChanged	Occurs when highlighted item is changed
SelectedItemChanged	Occurs when selected item is changed
RemoteKeyPress	Occurs in response to an user keystroke input
ScrollingFinish	Occurs when horizontal scrolling completed
VertScrollingFinish	Occurs when vertical scrolling between list items completed

3.5.3.1 HighlightedItemChanged

Syntax: HighlightedItemChanged (Byte index)

Description: This event occurs when the highlighted item is changed, that is, when the user moves through the list with the arrow keys.

index: Contains the index of the item which highlight has changed.

Code window usage example:

```
public void Load()
{
  osdListbox1.ItemText[0] = "HDMI 1";
  osdListbox1.ItemText[1] = "HDMI 2";
  osdListbox1.ItemText[2] = "HDMI 3";
  osdListbox1.ItemText[3] = "HDMI 4";

  OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1); }
```

```
private void osdListbox1_HighlightedItemChanged(Byte index)
{
      osdListbox1.FocusIndex = 2
   }
```

3.5.3.2 SelectedItemChanged

Syntax: SelectedItemChanged (Byte index, Boolean newStatus)

This event is triggered once when the selected new item is changed, that is, when the user sends the spacebar key.

index: Contains the index of the item which selection just changed.

newStatus: 0 if unselected, 1 if selected.

Code window usage example:

```
public void Load()
{
    osdListbox1.ItemText[0] = "HDMI 1";
    osdListbox1.ItemText[1] = "HDMI 2";
    osdListbox1.ItemText[2] = "HDMI 3";
    osdListbox1.ItemText[3] = "HDMI 4";

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
}

private void osdListbox1_SelectedItemChanged(Byte index, Boolean
  newStatus)
{
        osdListbox1.ScrollingEnabled = TRUE;

}
```

3.5.3.3 RemoteKeyPress

Syntax: RemoteKeyPress (Byte *keyCode, Boolean *cancel)

Description: This event occurs when the component has focus and the user presses and releases a key.

***keycode:** Pointer to the variable that stores the pressed key. User can modify its value to manually assign a different keycode to the keystroke.

***cancel:** Pointer to the variable the determines if the pressed key is further processed or not. If set to "false", the component will not receive the keystroke. For example, an OSDListbox will not receive the "down" key.

There are keys that automatically interact with the OSDListbox component without the need to define any code within the RemoteKeyPress event method. These keys are said to be hardcoded within the component, and are shown in Table 11.

Whenever the user presses and releases any key, the code execution jumps to the RemoteKeyPress event (if available). Then, if the keystroke was not modified or cancelled (*cancel = true), and it is one of the hardcoded keys, the keystroke is sent to the component, which reacts accordingly. For more information on the flow of the pressed key, refer to the Components section on page 14.

We could use right and left arrow keys to enable/disable other submenus

Code window usage example:

```
private void osdListbox1_RemoteKeyPress(Byte *keyCode, Boolean
*cancel)
{
    if (osdListbox1.FocusIndex == 0)
      {
       if (*keyCode == KEYBOARD_RIGHT) //Right Arrow => Show Submenu
        {
           osdListbox1.Visible = False;
           osdListbox_Submenu.Visible = True;

 OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox_Submenu);
      }
}
}
```

3.5.3.4 VertScrollingFinish

Description: This event occurs when the vertical scrolling between list items is complete.

3.5.3.5 ScrollingFinish

Syntax: ScrollingFinish(void)

Description: This event occurs when the horizontal text scrolling of one item is completed. Note that the event triggers just at the moment the text stops scrolling, that is, before waiting for the period defined by *ScrollingIntervalTime.*

3.6 OSDImage and OsdTboxImage

It is used for containing an image. The*OSDImage and OsdTboxImage* component can be used to display static images and animations (which are composed of tiles or frames).This component cannot receive focus, and has one associated event.

3.6.1 OSDImage and OsdTboxImage Properties

The property of the OsdImage and OsdTboxImage are identical.But OsdImge is drawn in hardware using Iboxes and OsdTboxImage is drawn in hardware using Tboxes.

Table 14 OSDImage and OsdTboxImage common animation properties

Property or Method	Short Description
AnimationEnabled	Sets if animation will be active
AnimationSpeed	Sets the animation speed in frames per second
AnimationType	Sets if the animation will loop continuously, bounce or run only once
ColorTable	Sets the color table to be used for the image
Image	Sets image to be displayed and set up for animation
InitialFrame	Sets the initial frame from the list of images when animation is enabled. (Starts at index 0)
FinalFrame	Sets the final frame from the list of image when animation is enabled.
CurrentFrame	Sets the starting frame to begin the animation with when animatino is enabled.

3.6.1.1 AnimationEnabled

Deactivate: Sets if the animation will be active.

Deactivate: True or false

Auto complete: Yes

Property declaration:

```
public bool Enabled{get; set;}
```

Start animating the osdImage1 component

Code window usage example:

```
osdImage1.Enabled = true;
```

3.6.1.2 AnimationSpeed

Description: Sets the animation speed in frames per second. The speed is frames interval between each tile. This is usually set through the GUI although it may be changed at run time.

Range: 0 – 1000

Property declaration:

```
public int Speed{get; set;}
```

Code window usage example:

```
osdImage1.Speed = 2;
```

3.6.1.3 AnimationType

Description: Sets whether the animation will loop, bounce or only one time. See the description of the OSDAnimationType global enumeration to find out the available options here. This is usually set through the GUI although it may be changed at run time. The section 4.4 in Blimp user manual shows an example for creating image animation.

Selection: Loop, Bounce, Only once

Property declaration:

```
public OSD_ANIMATIONTYPE AnimationType{get; set;}
```

Code window usage example:

```
osdImage1.AnimationType = OSD_ANIMATIONTYPE.LOOP;
```

3.6.1.4 ColorTable

Description: Sets the color table to be used for the image. The ADV7625 can store maximum 32 unique colors at any time for displaying the image for OsdImage. Maximum 8 unique colors is supported for OsdTboxImage.
Select Property Navigator → ColorTable → Edit to create the color table on Image color table editor, as shown in Figure 4.

User can add a colors using “Add Color Table” option or extract the colors from image using “Add Image Color” option.

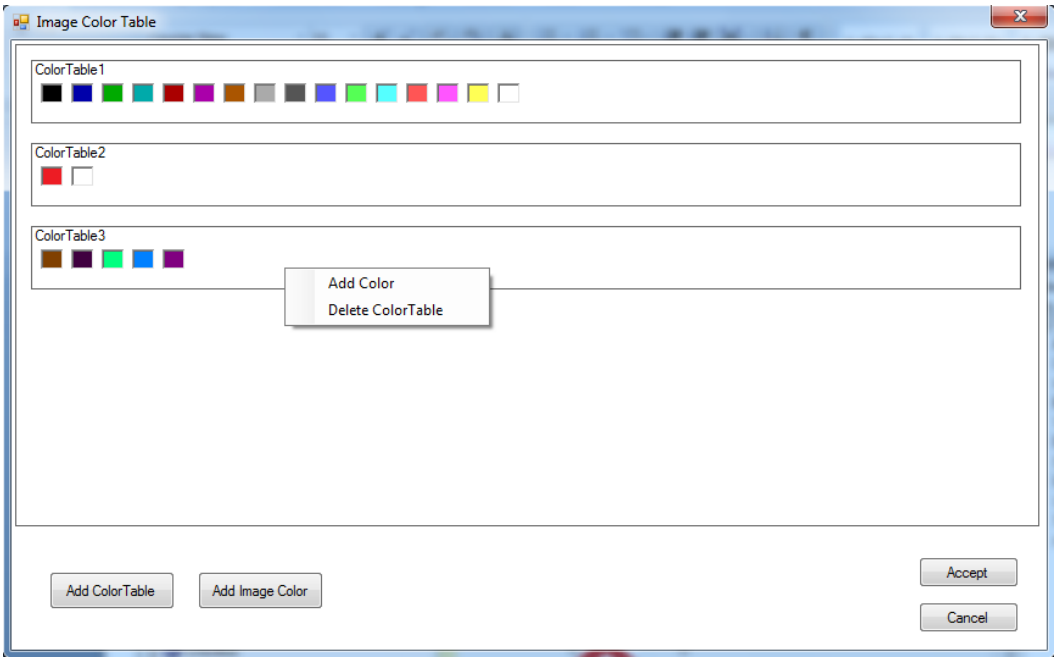


Figure 4: Image Color Table Editor

3.6.1.5 InitialFrame

Description: Sets the initial frame from the list of images when the animation is enabled. (Starts at index 0.)

Value: <= FinalFrame

Autocomplete: Available in code window for runtime modification. Can be read or written to.

3.6.1.6 FinalFrame

Description: Sets the final frame from the list of images when the animation is enabled.

Value: <= number of Frame images - 1

Autocomplete: Available in code window for runtime modification. Can be read or written to.

3.6.1.7 CurrentFrame

Description: Sets the starting frame to begin the animation when the animation is enabled.

Value: <= number of Frame images - 1

Autocomplete: Available in code window for runtime modification. Can be read or written to.

3.6.2 OSDImage Events

Table 15OSDImage Events

Event	Short Description
EnableChanged	Occurs when the Enabled property for animation has changed

3.6.2.1 EnableChanged

Syntax:EnableChanged (Boolean)

This event occurs when the Enabled property for animation has changed, whether it is the user assigning a new value or due to the end of a current animation. This may be used to know when an animation has finished, for example, chain animations.

3.7 OSDProgressbar

The *OSDProgressbar* works by overlapping three Fboxes over one in the background Fbox, second in the outline Fbox and third in the bar Fbox. Depending on the position set to the barvalue. This component can receive focus and has two events associated with RemoteKeyPress and ValueChanged.

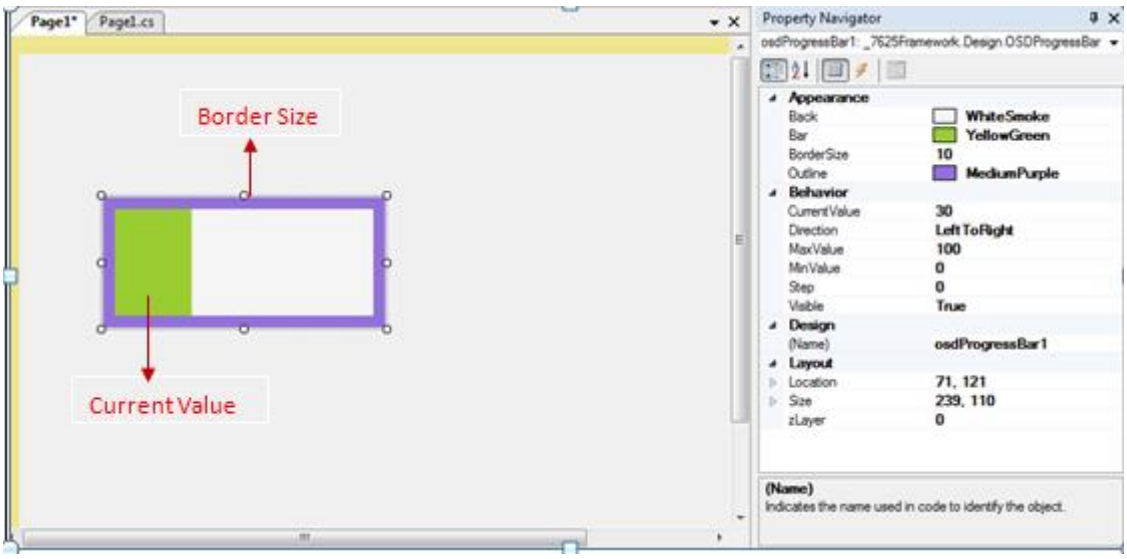


Figure 5:OSDProgressbar on canvas

3.7.1 OSDProgressbar Properties

Table 16OSDProgressbar Properties

Property or Method	Short Description
MinValue	Sets the lower bound value.
MaxValue	Sets the upper bound value.
• **Step** • **addStep()** • **subtractStep()**	Sets the value which the progress bar will increment and decrement. Adds the amount defined in the property Step to the current position. Subtracts the amount defined in the property Step to the current position.
CurrentValue	Sets the value of the progress bar. This value must be between MinValue and MaxValue
Color • **Bar** • **Outline** • **Back**	 Sets color of progress bar Sets color of progress bar outline border Sets the color of the background bar
BorderSize	Sets border width of all sides
Direction	Sets progress bar filling direction (horizontal or vertical)

3.7.1.1 MinValue

Description: Sets the lower bound value. This is the minimum value that the bar can reach, even if the SubtractStep() method is being called.

Units: 16-bit integer

Property declaration:

```
public short MinValue{get; set;}
```

Code window usage example:

```
Min value for contrast bar set to 0
contrastBar.MinValue = 0;
```

3.7.1.2 Max Value

Description: Sets the upper bound value. This is the maximum value that the bar can reach, even if the AddStep() method is being called.

Property declaration:

```
public short MaxValue{get; set;}
```

Max value for contrast bar set to 100

Code window usage example:

```
contrastBar.MaxValue = 100;
```

3.7.1.3 Step

Description: Sets the value which the progress bar will increment and decrement for each step. This value will be added or subtracted from the foreground image when calling AddStep() or SubtractStep() methods, or when using the hardcoded keys of the component.

Units: 8-bit integer

Property declaration:

```
public short Step{get; set;}
```

Code window usage example:

```
contrastBar.Step = 1;
```

3.7.1.4 addStep()

Description: Add the amount defined in the property *Step* to the current position.

Property declaration:

```
public UINT32 addStep()
```

Make the bar to grow up

Code window usage example:

```
contrastBar.addstep();
```

3.7.1.5 subtractStep()

Description: Subtracts the amount defined in the property *Step* to the current position.

Property declaration:

```
public UINT32 subtractStep()
```

Make the bar to shorten

Code window usage example:

```
contrastBar.subtractstep();
```

3.7.1.6 CurrentValue

Description: Sets the value of the progress bar. This is useful to set the default value for the bars, before the user can edit it. This value must be between *MinValue* and *MaxValue*.

Property declaration:

```
public short Position{get; set;}
```

Contrast bar set to 50 by default

Code window usage example:

```
contrastBar.Position = 50;
```

3.7.1.7 Color

Description: Sets the colors of the progress bar.

Back

Description: Sets the color of the background bar.

Range: See RGB color selection entry limit

Auto complete: No

Bar

Description: Sets the color of the progress bar.

Auto complete: No

Outline

Description: Sets the color of the progress bar outline border.

Auto complete: No

3.7.1.8 BorderSize

Description: Sets the border width for all sides.

Units: Pixels

Range: Top, Bottom, Left

Auto complete: No

3.7.1.9 Direction

Description: Sets the progress bar filling direction (LeftToRight or RightToLeftl).

Selection: refer the Directionsection for select a direction of OSDProgressBar component.

Auto complete: No

3.7.2 OSDProgressbar Events

Table 17OSDProgressbar Events

Property or Method	Short Description
ValueChanged	Occurs whenever the length of the bar gets modified
RemoteKeyPress	Occurs in response to a user keystroke input

3.7.2.1 ValueChanged ()

This event occurs whenever the length of the bar gets modified.

3.7.2.2 RemoteKeyPress

Syntax: RemoteKeyPress (Byte *keyCode, Boolean *cancel)

This event occurs when the component has focus and the user presses and releases a key

***keycode:** Pointer to the variable which stores the pressed key. User can modify its value to manually assign a different keycode to the keystroke.

***cancel**: Pointer to the variable which determines if the pressed key is further processed or not. If set to "false", the component does not receive the keystroke (for example, an OSDProgressBar will not receive the "right" key).

There are keys that automatically interact with the OSDProgressBar component, without the need to define any code within the RemoteKeyPress event method. These keys are said to be hardcoded within the component, and are shown inTable 11. Whenever the user presses and releases any key, the code execution jumps to the*RemoteKeyPress* event (if available). Then, if the keystroke was not modified or cancelled *(*cancel = true*), and it is one of the hardcoded keys, the keystroke is sent to the component, which reacts accordingly. For more information on the flow of the pressed key, refer to the Pressed Key Event Flow Between Pages and Components section.

Note that when using the hardcoded keys, the OSDProgressbar is filled/emptied by an amount equal to the unit defined in the *Step* property.

Table 18Hardcoded Keys in OSDProgressbar Component

Key	Key Code (decimal)	Function
Left Arrow	37	Empty bar (only available when in horizontal orientation)
Right Arrow	39	Fill bar (only available when in horizontal orientation)
Up Arrow	38	Empty bar (only available when in vertical orientation)
Down Arrow	40	Fill bar (only available when in vertical orientation)

3.8 OSDHistogram

The *OSDHistogram* allows displaying tabulated frequencies shown as bars (bins). It can be used to display preset audio equalizations or in run time to display the spectral content of the audio being played. It cannot have focus set to it and it does not have any event associated to it.

3.8.1 OSDHistogram Properties

Table 19OSDHistogram Properties

Property or Method	Short Description
BinColor	Sets bins color
MaximumValue	Sets maximum value for range of each bar
MinimumValue	Sets minimum value for range of each bar
Orientation	Sets orientation (vertical or horizontal) of component
Spacing	Sets spacing (in pixels) between bars
TotalBins	Sets total number of bars in which histogram is divided
setBinValue()	sets the value for an individual bin specified by the index.
setBinsValue()	sets the value for multiple bins as specified by beginning and end index.

3.8.1.1 BinColor

Description: Sets the bins color.

Selection: See RGB color selection entry limit

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
publicuint BinColor {get; set;}
```

Code window usage example:

```
osdHistogram1.BinColor = 0xFF00FF00u;
```

3.8.1.2 MaximumValue

Description: Sets the maximum value for the range of each bar.

Units: 16-bit integer

Autocomplete: Available in code window for runtime modification. Can be read or written to. This property can be changed at runtime.

Property declaration:

```
public short MaxValue {get; set;}
```

Code window usage example:

```
osdHistogram1.MaxValue = 10;
```

3.8.1.3 MinimumValue

Description: Sets the minimum value each individual bar can reach.

Units: 16-bit integer

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public short MinValue {get; set;}
```

Code window usage example:

```
osdHistogram1.MinValue = 0;
```

3.8.1.4 Orientation

Description: Sets the orientation (vertical or horizontal) of the component.

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public OSD_ORIENTATIONOrientation {get; set;}
```

Code window usage example:

```
osdHistogram1.Orientation = OSD_ORIENTATION.Horizontal;
```

3.8.1.5 Spacing

Description: Sets the spacing (in pixels) between the bars.

Units: 8-bit integer

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public byte Spacing {get; set;}
```

Code window usage example:

```
osdHistogram1.Spacing = 1;
```

3.8.1.6 TotalBins

Description: Sets the total number of bars in which the histogram is divided. Maximum value is 255 bars.

Autocomplete: Available in code window for runtime modification. Can be read or written to.

Property declaration:

```
public byte TotalBins {get; set;}
```

Code window usage example:

```
osdHistogram1.TotalBins = 10;
```

3.8.1.7 setBinValue()

Description: This method sets the value for each individual bin specified by the index..Bars which are not set get a default value of 0. Bars which are assigned a value bigger or smaller than the one set, respectively, through *MaxValue* or *MinValue*, are also assigned a value of 0. Note that the first bin is number 0, so the index runs from 0 to *TotalBins* – 1.

Property declaration:

```
public UINT32 setBinValue(UINT8 index,INT16 value);
```

Code window usage example:

```
osdHistogram1.TotalBins = 10;
osdHistogram1.MaxValue = 10;
osdHistogram1.MinValue = 0;
osdHistogram1.Spacing = 1;

for (i = 0; i < 10; i++)
{
   osdHistogram1.setBinValue(i,(short)i);
}
```

Due to hardware limit of 10 Fbox per video line, there is a limit of 10 vertical bins in histogram

3.8.1.8 setBinsValue()

Sets the value for multiple bins as specified by the beginning and end index.

Method declaration:

```
UINT32 setBinsValue(UINT8 numOfBins, UINT8 startingBin,

INT16[] binValueArray )
```

Code window usage example:

```
short[] value_buffer = new short[5];

value_buffer[0] =10;
value_buffer[1] =20;
value_buffer[2] =30;
value_buffer[3] =40;
value_buffer[4] =50;
osdHistogram1.MaxValue =100;
osdHistogram1.MinValue = 0;
osdHistogram1.TotalBins = 5;
osdHistogram1.setBinsValue((byte)5, (byte)0, value_buffer);
```

3.9 OSDBox

The OSDBox used to draw the box with frame effect.

3.9.1 OSDBox Properties

Table 20OSDBox Properties

Property or Method	Short Description
Color	Sets the box fill color
BorderColor	Sets the box border color in RGB format
Border • **LeftBorder** • **RightBorder** • **TopBorder** • **BottomBorder**	 Sets if left border is displayed Sets if right border is displayed Sets if top border is displayed Sets if bottom border is displayed

3.9.1.1 Color

Sets the box fill color in RGB format.

Property declaration:

```
public ColorColor{get; set;}
```

Set a box fill color to Red color

Code window usage example:

```
osdBox1.Color = 0xffff0000u;
```

3.9.1.2 BorderColor

Sets the box fill border in RGB format.

Property declaration:

```
public ColorBorderColor{get; set;}
```

Set a box border color to Green color

Code window usage example:

```
osdBox1.BorderColor= 0xff00ff00u;
```

3.9.1.3 LeftBorder

Sets if left border is displayed

Property declaration:

```
public bool LeftBorder{get; set;}
```

Code window usage example:

```
osdBox1.LeftBorder = TRUE;
```

3.9.1.4 RightBorder

Sets if right border is displayed

Property declaration:

```
public bool RightBorder{get; set;}
```

Code window usage example:

```
osdBox1.RightBorder = TRUE;
```

3.9.1.5 TopBorder

Sets if top border is displayed

Property declaration:

```
public bool TopBorder{get; set;}
```

Code window usage example:

```
osdBox1.TopBorder = TRUE;
```

3.9.1.6 BottomBorder

Sets if bottom border is displayed

Property declaration:

```
public bool BottomBorder{get; set;}
```

Code window usage example:

```
osdBox1.BottomBorder = TRUE;
```

4 Project Settings

To modify parameters, select *Project →Show project settings*, and the tab shown in Figure6 is displayed.

4.1 Design

4.1.1 Designer layout

Snap lines*:* Enables components to snap on grid lines on the designer canvas.**Grid***:* Enables grid to be visible on designer canvas.

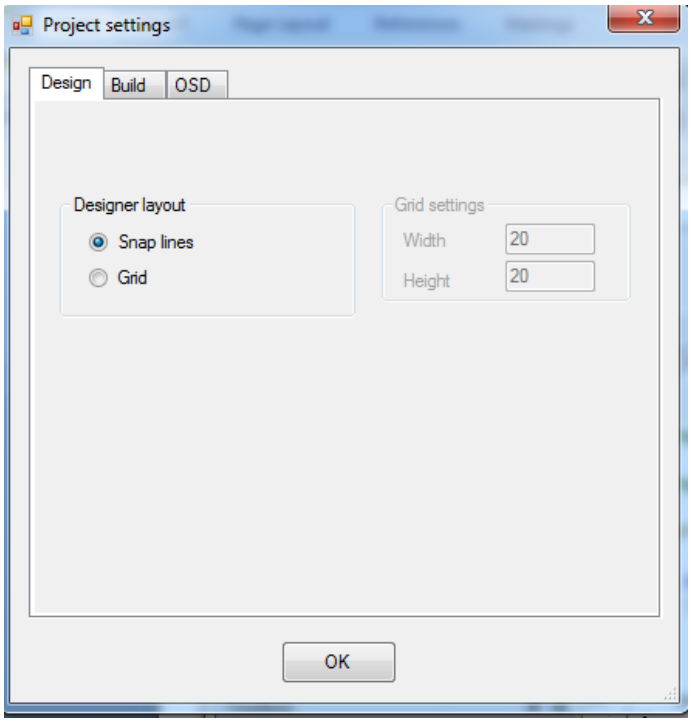


Figure6: Designer Layout Tab

4.2 Build

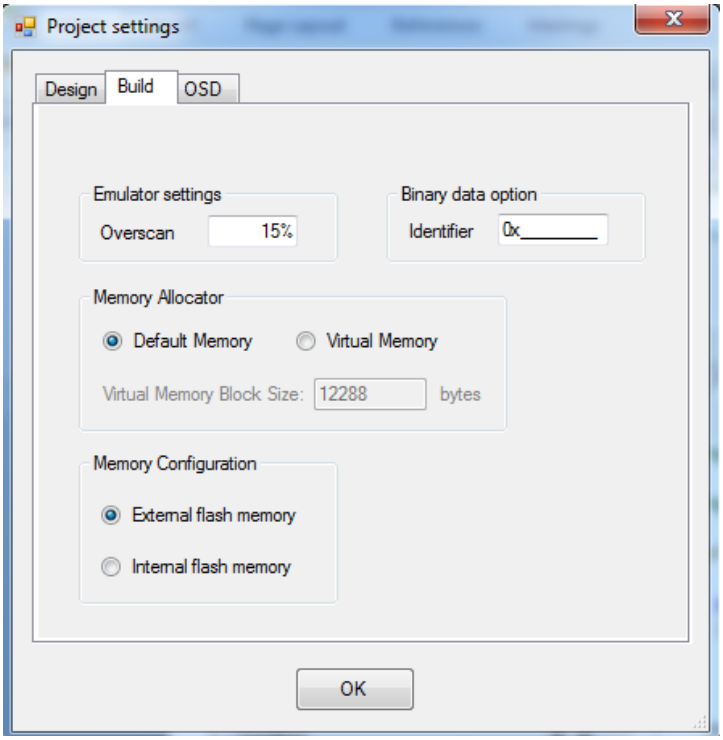


Figure7: Build Layout Tab

4.2.1 Emulator overscan

Allows setting a percentage of overscan, which will be presented as dark areas on the emulator window. Note that this setting only provides an impression of how the OSD might be cropped when shown on TV; it does not have any effect on the designed OSD apart from the view shown in the emulator window.

4.2.2 Binary data option

Appends the binary identifier to the flash binary data file. This option is used if flash contains multiple data block either for 2 separate OSD design or other applications.

4.2.3 Memory Allocator

Default memory: The firmware will use ANSI-C memory allocation function to allocate memory when enabling and disabling components and pages. The MCU compiler heap will be used for the memory.

Virtual memory: A static memory block is defined and the firmware will use custom functions to allocate memory from the virtual memory block defined. The MCU compiler stack is used and the block is assign at program initialization.

Virtual Memory Block Size: Defines the size for the virtual memory that will be defined on the stack. If it is exceeded, a failure message will be logged and program will halt.

4.2.4 Memory Configuration

User can configure either external flash memory or internal flash memory. External flash memory: The OSD data's will be stored in ddr2_dump_raw.bin file under the release folder. The binary file needs to be flashed using Blimp tool. Internal flash memory: The OSD data's will be stored in data memory. No need to flash the binary file.

4.3 OSD

4.3.1 Resolutions

Under the OSD settings tabs, the user can define the OSD resolutions which each page can be defined for. Each page will can define unique position and size for all OSD components and a set of scaling for Fbox, Tbox and Ibox will which be reflected on the designer canvas. See section 5 for more details

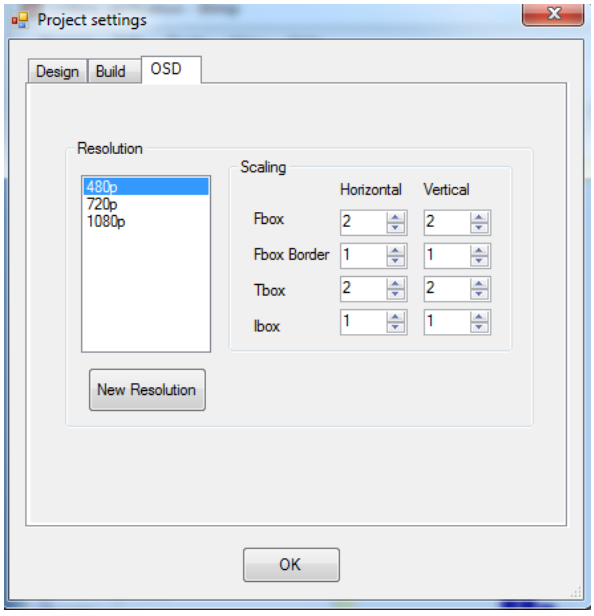


Figure8: OSD Resolution Tab

5 Resolution configuration

The resolution definition are required because the integer scaling factor is not able to align the text position and size with boxes and images after scaling and most cases, the scaling wanted is not an exact integer between resolutions. This is mostly important for Tbox and Ibox since their size must be a scale factor of two if same fonts and stored images are to be used and so position is usually adjusted to fit proportionally as close as possible for all resolutions. For Fbox both the size and position are usually adjusted to fit proportionally as close as possible for all resolutions.

The below table shows an example of scaling factor, which will be applied to position and size in copy resolution setting.

	480p	**720p**	**1080p**	**4K2K**
Resolution (Width x Height)	720x480	1280x720	1920x1080	1920x1080
Scalar Factor	1:1	1.78:1.5	2.67:2.25	2.67:2.25
Component Location / Size (Width x Height)	20x10	36x15	53x22	53x22

5.1 Specific 4K2K Page

User can able to be able to display the OSD with 4k2k resolution as like other OSD Resolution. The project settings as shown in below figure will be the option for designing the same.

The spec of 4K2K resolution is mainly **3,840 X 2,160**. Compared to Full HD, which has a resolution of **1,920 X 1,080**, the resolution of 4K2K is 4 times (2 times Horizontal x 2 times Vertical) higher than Full HD.In hardware, for 4K2K resolution there is a limitation for the position display (Tbox & Ibox have only 10 bits, so it will allow maximum value of 1024 and Fbox has no limitation) So the position register bit is enabled for Tbox & Ibox and User can copy OSD for 4k2k resolution as like other resolutions with some changes done automatically in Blimp side in order to avoid this hardware limitation.

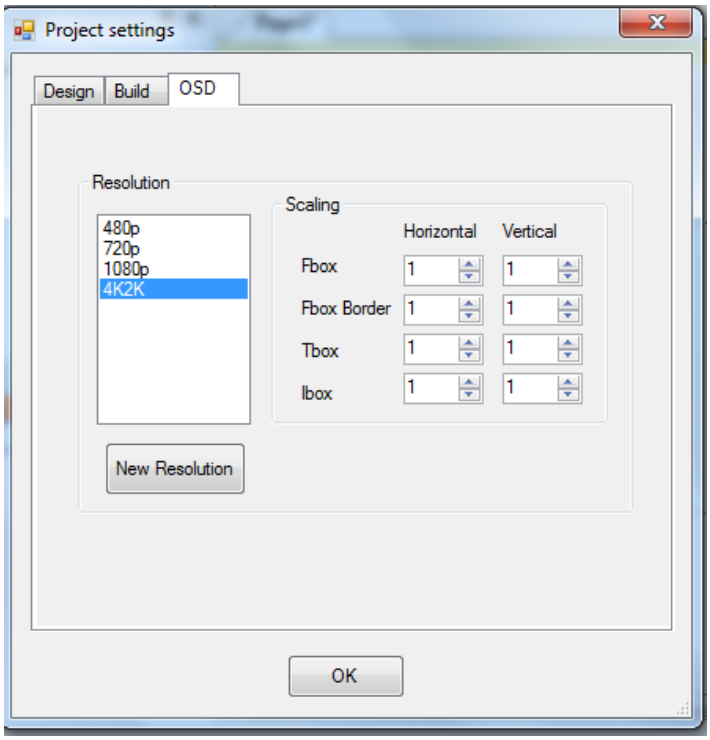


Figure9: OSD Resolution Tab

Vertical scaling is doubled in firmware because horizontal scalar register bit causes horizontal scaling for all components. Location property will apply scaling as specified by customer but it will round value so it is a multiple of the scaling for x and y. Location X in Blimp property must be multiple of horizontal scaling value. Location Y in Blimp property must be multiple of vertical scaling value (which is doubled in this case).Value stored in OSD RAM for position of Ibox and Tbox must be divided by scaling value. When user tries to enter a value non-multiple of scaling value in Location, the value will be automatically rounded to nearest.

For sample, we assume that we have OSD design page of 480p resolution which has a component location of X, Y is **100,100** with Horizontal scalar Integer value of **2** and vertical scalar Integer value of **2**. We may expect the component Location would be **267,225** with Scalar integer value of **6, 6** for the 1080p resolution. We also expect the Location would be **534,450** with scalar Integer value of 12, 12 for 4K2K Resolution, because 4K2K resolution is just 2 times Horizontal and Vertical values of 1080p resolution. But as we mentioned earlier, because of the hardware limitation in 4K2K page, we are not able to display the calculated values in canvas window. So we are displaying it in the adjusted nearest possible value as **528,444** instead of **534,450**.

5.2 Copy Resolution

User can copy the resolution settings (location & size) from designed resolution to the active resolution using the “Copy Resolution” icon in menu bar withour re-designed.

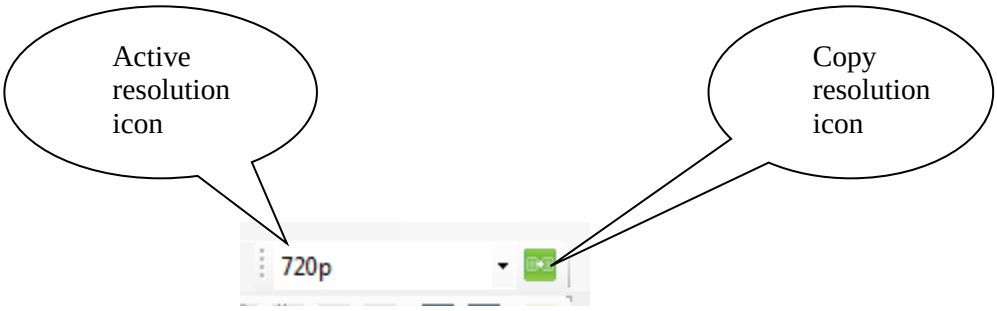
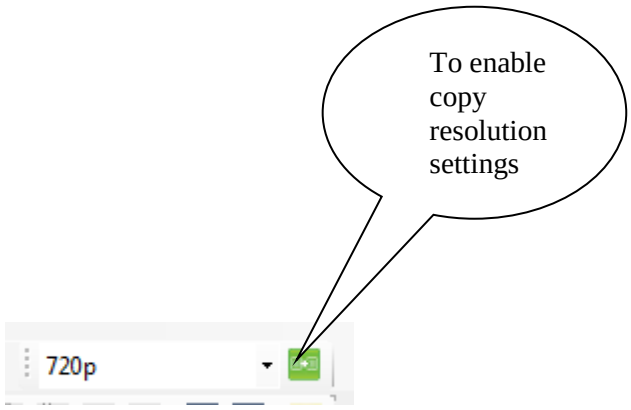


Figure 10: Active resolution on menu bar

Copy Resolution Settings:

Horizontal and vertical scalar values are editable; the active component settings (location and size) are updated automatically after confirmation.



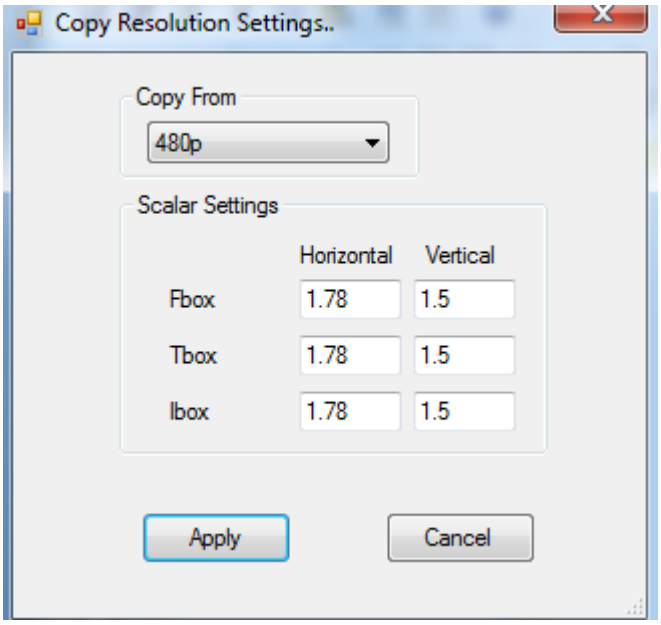


Figure 11: Copy Resolution Settings

Current resolution's all components of location and size would be replaced by reference resolution data.

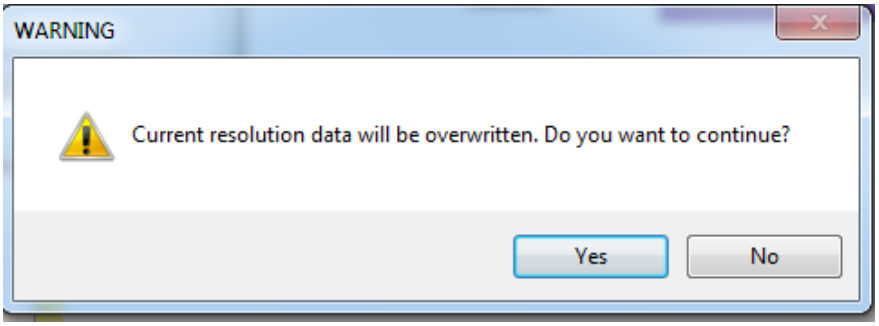


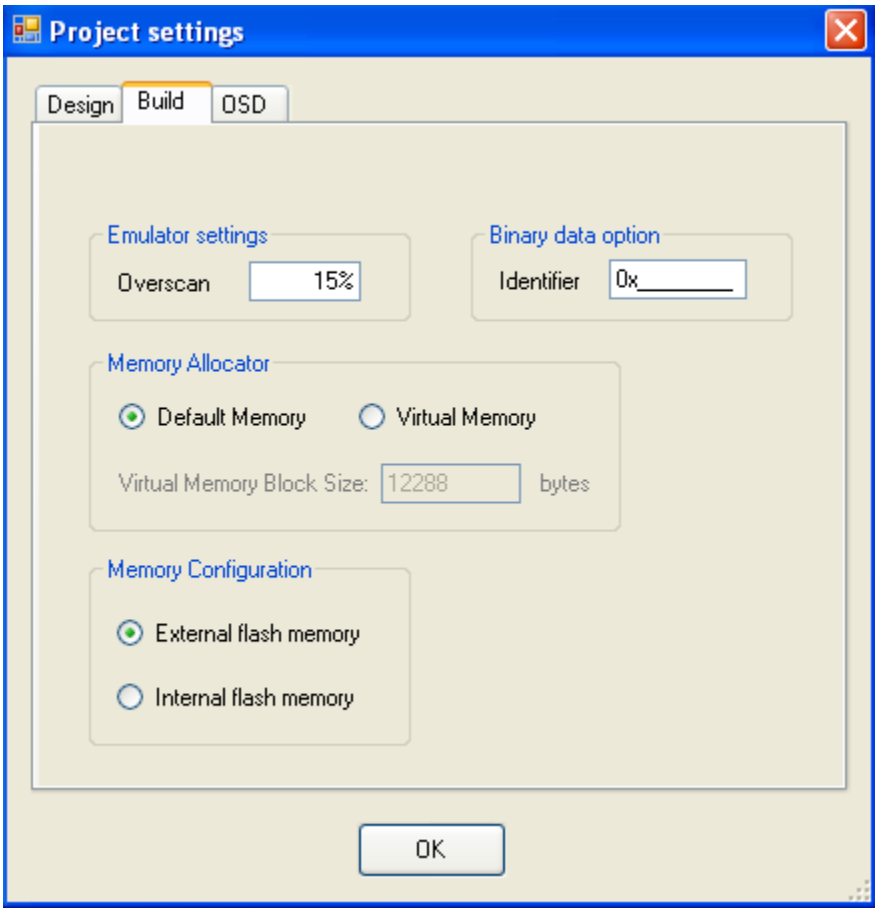
Figure 12: Confirmation windows for applying copy resolution data

6 Memory management

Memory management is to provide ways to dynamically allocate portions of *memory* to data at its request, and free it for reuse when no longer needed.

Memory Allocator:

Memory Allocator settings includes two options 1) Default Memory 2) Virtual Memory



In default memory, the given data will be allocated in dynamic using Local_malloc API.

Local_malloc API in framework is used to allocate a heap memory inside the function temporarily. This memory is de-allocated using Local_Free API before the end of that particular function.

But in some cases, when data is allocated in the memory using Local_malloc for whole project, it will be deallocated using Local_Free API before reallocation of some other data (Example: Text in OSDLabel).

In Virtual memory, the given data will be allocated in sequence using Memory Allocator_malloc API.

The virtual memory can be used to allocate the memory for page and component creation in blimp generated files.

7 ADV7625 Static OSD Framework

7.1 Introduction

The new framework selection is available to create a static page OSD with ADV7625 in master mode. The option is called "ADV7625 Static OSD framework" when creating project. When setting the ADV7625 in master mode, the OSD data is read directly from flash to ADV7625 OSD RAM.

With this framework, all pages are static meaning that each page is stored in the flash and read as is with no animation or text scrolling features.

The framework makes the code size in MCU much smaller. The Blimp generated code only generates API to command ADV7625 to read data from flash.

RAM requirements are much lower since it does not manipulate data from flash

7.2 Using OSD Components

7.2.1 Page

OsdPage properties are implemented in 7625 static osd framework as like 7625 framework except page events

7.2.2 Components

The components currently supported for the static page framework are:

- OsdBox
- OsdProgressBar
- OsdHistogram
- OsdLabel
- OsdTboxImage
- OsdImage

All components properties are implemented as like 7625 except components events

7.3 Project Settings for ADV7625 Static OSD

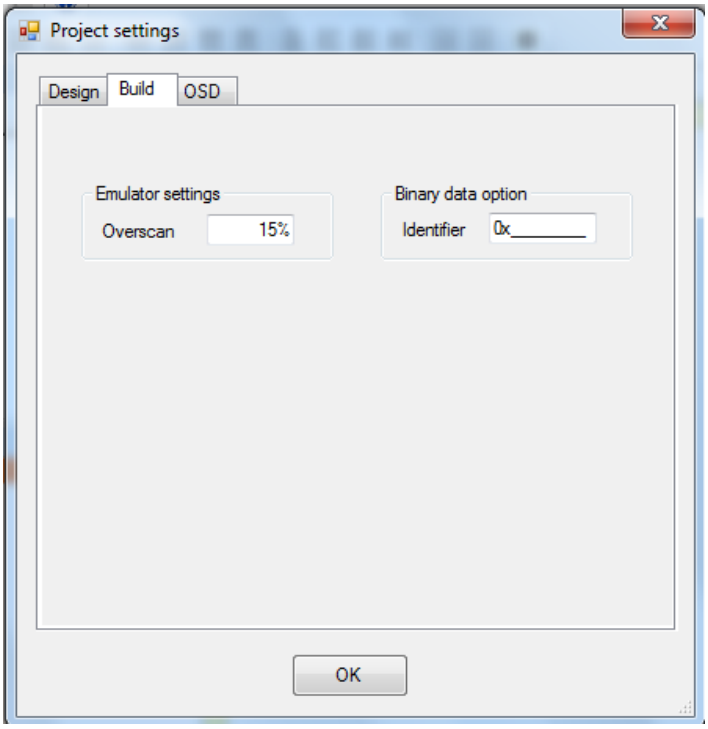
To modify parameters, select *Project →Show project settings*

7.3.1 Design

Designer layout is implemented as like 7625 framework for 7625 Static OSD framework.

7.3.2 Build

Setting Emulator over scan and Binary data option in ADV7625 static OSD framework as like ADV7625 framework



Build Layout Tab

7.3.3 OSD (Scalar Settings)

Scalar settings for each resolution as like 7625 framework

7.4 External flash structure

No of Unicode in project **Block size is 2 byte**
No of icon in project **Block size is 1 byte**
No of Fbox color **Block size is 1 byte**
No of Tbox color **Block size is 1 byte**
No of Ibox color **Block size is 1 byte**
No of Fbox **Block size is 1 byte**
No of Tbox **Block size is 1 byte**
No of Ibox **Block size is 1 byte**
Font glyph information Each font glyph is 16x16 pixels and required 33 bytes and maximum 256 font glyph can be stored in Font RAM **Block size is 33 byte**
Icon data Information Each icon occupies 8x8 pixels and required 40bytes, maximum 64 icon is supported **Block size is 40 byte**
Fbox color& back ground color information Each Fbox color takes 6 bytes and maximum 8 color is supported **Block size is 6 byte**
Tbox color information Each Tbox color takes color takes 3 bytes and maximum 16 color is supported **Block size is 3 byte**
Ibox color information Each Tbox color takes color takes 3bytes and maximum 32 color is supported **Block size is 3 byte**
Fbox unit information Fbox location, size, priority **Block size is 7 byte**
Fbox configuration information Fbox unit is linked with Fbox color

Block size is 1 byte
Tbox unit information Tbox location, priority **Block size is 4 byte**
Tbox configuration information Tbox unit is linked with Tbox color **Block size is 1 byte**
Ibox unit information Ibox location, priority, icon handle **Block size is 5 byte**

NOTES