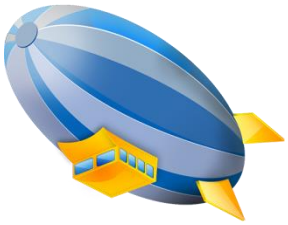


Blimp

OSD development tool

By ANALOG DEVICES

ADV800x Framework User Guide

Preliminary 0.15

Advantiv
Advanced Television Solutions
by Analog Devices

ANALOG DEVICES

ADV800x Framework Manual for 3.9Rel
December 2015

Copyright Information

Disclaimer

Trademark and Service Mark Notice

Contents

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

The product features:

Easy to use GUI to design On Screen Display (OSD)menus for the ADV series with OSD
Wide range of OSD graphic components available to create state of the art OSD
Multi language configuration and Unicode character map selection
Simple integration of output generated code and ready use compressed flash file

Purpose of This Manual

The ADV8003Blimp Framework User Manual provides detailed description of each components and functionalities in Blimp specific to ADV8002 framework.

Intended Audience

The primary audience for this manual are OSD Designers and software developers creating OSD designs using Blimp tool.

Technical or Customer Support

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

Visit the Analog Devices' Analog DVI and HDMI interfaces product web site at
http://www.analog.com/en/audiovideo-products/analoghdmidvi-interfaces/products/index.html
Post your questions in EngineerZone support: http://ez.analog.com/community/video
E-mail tools questions to
processor.tools.support@analog.com
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.

EngineerZone

EngineerZone 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.

Social Networking Web Sites

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

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

1 OSD framework overview

1.1 Introduction

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

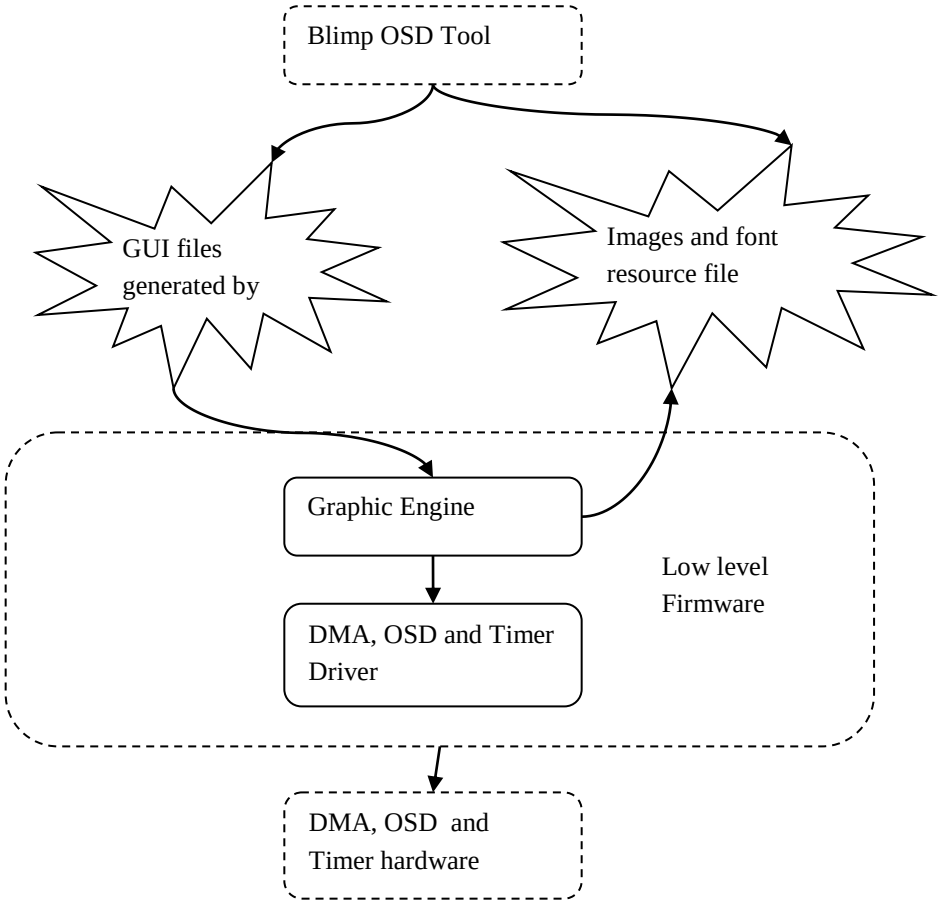


Figure 1: Block Diagram of low level firmware

Figure 2 displays the flow for using firmware.

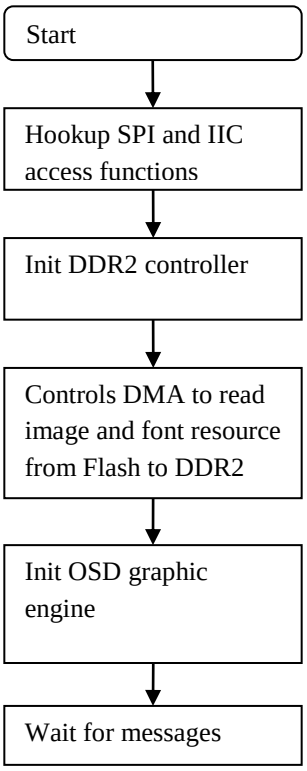


Figure 2:Flow for using low level firmware

In the step of initializing OSD engine, just one API will be called, inside which the firmware will register components, initialize hardware and call GUI source files generated by Blimp OSD tool to create GUI and map messages/events. As the firmware is message/event driven, just send any message/event, including keypad input and infrared remote controller input, to firmware current focus window then GUI will operate as same as customized in Blimp OSD tool.

In the following sections the details of graphic engine and hardware driver will be presented.

1.2 Graphic engine

1.2.1 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 all components that are already implemented in firmware.

Table 1: Implemented OSD Components in Firmware

OSD Component	Description
OSD_LABEL	Used to display text information or notice within the OSD
OSD_LISTBOX	Displays a list of user-selectable options
OSD_IMAGE	Container for an image or animation
OSD_PROGRESSBAR	Displays a progress bar
OSD_MENUBAR	List of icons which the user can navigate through
OSD_TEXTBOX	Allows the user to insert text
OSD_IPTEXTBOX	Allows the user to insert an IP address
OSD_KEYBOARD	Displays a virtual keyboard on the screen
OSD_HISTOGRAM	Displays a graphic equalization
OSD_TIMER	Triggers an event at user-defined interval
OSD_MULTICOLUMNLISTBOX	Displays a list of user-selectable options with icons, text and Slider
OSD_BOX	Displays a box with border
OSD_CIRCLE	Displays a circle
OSD_LINE	Displays a line
OSD_SLIDER	Displays a slider with bar
OSD_ICONLISTBOX	Displays a list of user-selectable options with single icon and text
OSD_PICTUREBOX	Displays a picture box
EXTERNALOSD	Displays an External OSD video

1.2.2 Components and Regions

OSD hardware uses region to display an area of DDR2 memory and could support max 256 regions to display at the same time. The number that could be displayed at the same time depends region overlapping status, region size, etc..

For each region, the corresponding hardware registers include the following information:

Position

Size

Effects

Color depth (8-bits, 16-bits or 24-bits each pixel)

Address in DDR2 memory

AnimationInformation

Enable

Z value

Alpha

Not all components are directly mapped to one or more regions. Relationship between component and region is shown in Table 2.

Table2: Map between component and region

OSD Component	Region Number
OSD_LABEL	1
OSD_LISTBOX	1
OSD_IMAGE	1
OSD_PROGRESSBAR	2
OSD_MENUBAR	Dynamic (depends on nodes number on current level)
OSD_TEXTBOX	1
OSD_IPTEXTBOX	1
OSD_KEYBOARD	2
OSD_HISTOGRAM	1
OSD_TIMER	0
OSD_MULTICOLUMNLISTBOX	1
OSD_CIRCLE	1
OSD_BOX	1
EXTERNALOSD	1
OSD_LINE	Dynamic (1 region in DDR2 memory)
OSD_SLIDER	Dynamic(4 regions in DDR2 memory)

OSD_ICONLISTBOX	1
OSD_PICTUREBOX	Dynamic (9 regions in DDR2 memory)

2 OSD API

This chapter provides information about pages in Blimp.

The following topics are covered:

“Pages” on page 32

2.1 Pages API

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.

2.2 APIs Show/Hide and Focus Pages

This section shows the methods which can be used to control visibility and focus of the pages within Blimp.

Table 2: Pages APIs

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
OSDEG_setPageZlayer	Sets Zlayer of the OSDPage

2.2.1 ADIAPI_OSDEgShowPage

Syntax: `public`void ADIAPI_OSDEgShowPage (HOBJ Page, UINT8 NotUsed);

Displays one page on the screen.

Code window usage example:

```
OsdApi.ADIAPI_OSDEgShowPage(PageManager.InputSelection, 0);
```

2.2.2 ADIAPI_OSDEgHidePage

Syntax: `public void ADIAPI_OSDEgHidePage(HOBJ Page, UNIT8 NotUsed);`

Hides one page from being displayed on the screen.

Code window usage example:

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

2.2.3 ADIAPI_OSDEgSetFocusPage

Syntax:`public void ADIAPI_OSDEgSetFocusPage(PageManager);`

Sets the focus to one page.

Code window usage example:

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

2.2.4 OSDEG_setPageZlayer

Sets zlayer of the OSD Page.

Method declaration:

```
public void OSDEG_setPageZlayer(IPage myPage, UINT8 z);
```

Parameters:

z:ZLayer value

Code window usage example:

```
OsdApi.OSDEG_setPageZlayer(PageManager.pMain, 0);
```

2.3 Other APIs

This section presents other APIs which can be used in Blimp pages and components.

Table 3: More Pages APIs

API	Short Description
ADIAPI_OSDEgSetLanguage	Sets language in use within OSD
OSDEG_ReadConstStringW	Reads constant string from DDR2 memory and places it into buffer
ADIAPI_OSDEgSetOSDBgColor	Sets background color of OSD

ADIAPI_OSDEgSetFocusComponent	Sets focus to a component
ADIAPI_OSDEgSendKeyComp	Sends key to any component
ADIAPI_OSDEgSendKeyPage	Sends key to any page
ADIAPI_OSDEgIsAnimationRunning	Determines if a component is running an animation
ADIAPI_OSDEgForceUpdate	Force OSD to update region data.

2.3.1 ADIAPI_OSDEgSetLanguage

Sets the language in use in the OSD.

Syntax: `public`void ADIAPI_OSDEgSetLanaguage(OSD_LANGUAGES language)

Code window usage example:

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

2.3.2 OSDEG_ReadConstStringW

Reads *bufferLength*characters from the *CONSTANT_STRING*constant string stored in DDR2 memory defined in the language *OSD_LANGUAGES.LANGUAGE*, and places it into *buffer*. Since the constant strings are Unicode by definition, the *buffer* needs to be declared as a 16-bit unsigned array. Note that since strings are null terminated, *buffer*should always be one character bigger than the constant string.

Method declaration:

```
public INT32 OSDEG_ReadConstStringW (ushort[] buffer, ushort size,
StringManager.CONSTANT_STRING stringId, OSD_LANGUAGES.LANGUAGE
language);
```

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) which are going to be read from the constant string. For example, if the constant string length is 11, *size* should be set to 12.

stringId: name of the constant strings as defined in the language window.

language: Language name as defined in the language window.

Returnvalue: Contain the length of the constant string if successful. If unsuccessful it will return '-1'. '-1' can also mean the allocated size in the buffer was too small to hold the whole string, for example, if the constant string is 11 characters long but *size* is 10.

Code window usage example:

```
private void ReadConsString()
{
    ushort[] buffer = new ushort[15];
    int returnValue;

    //CONTANT_TEXT constant string defined as "Hello World"
    osdLabel1.ConstText = StringManager.CONSTANT_TEXT;

    osdLabel2.Text = "Unitialized";
    osdLabel3.Text = "Unitialized";
 //CONTANT_TEXT constant string is 11 chars long but we need to reserve
12 chars in the buffer
    returnValue = OsdApi.OSDEG_ReadConstStringW(buffer, 12,
StringManager.CONSTANT_TEXT, OSD_LANGUAGES.DEFAULT);

    //osdLabel2 reads now "11".
    osdLabel2.setTextFormat("%d",returnValue);

    //osdLabel3 reads now "Hello World".
    osdLabel3.TextW = buffer;}
```

2.3.3 ADIAPI_OSDEgSetOSDBgColor

Sets the background color of the OSD

Method declaration:

```
publicvoid ADIAPI_OSDEgSetOSDBgColor (INT32 color);
```

Parameters:

color:32-bits value (A,R,G,B).

Code window usage example:

```
//Sets the color to full opaque red
ADIAPI_OSDEgSetOSDBgColor(0xFFFF0000);
```

2.3.4 ADIAPI_OSDEgSetFocusComponent

Sets the focus to one page component.

Method declaration:

```
publicvoidADIAPI_OSDEgSetFocusComponent (OSDControl Obj);
```

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

Code window usage example:

```
//Allow the user to move through an OSDListbox component called mainMenu
by setting focus to it

 OsdApi.ADIAPI_OSDEgSetFocusComponent(mainMenu);
```

2.3.5 ADIAPI_OSDEgSendKeyComp

Sends a key to any component without user intervention or without giving focus to the component.

Method declaration:

```
publicvoidADIAPI_OSDEgSendKeyComp(OSDControlmyControl,KeyCode keyCode);
```

Parameters:

myControl: Any component which can receive a key input.

***keyCode:**keyCode to be sent.

Code window usage example:

```
//When moving through a osdMainMenu component, it is possible to send
keystrokes to, for example, a //progress bar to create a kind of
scrolling bar

private void osdMainMenu_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{
  if(*keyCode == 0x26) //Up

    OsdApi.ADIAPI_OSDEgSendKeyComp(osdProgressbar1, 0x26);

  if(*keyCode == 0x28) //Down

    OsdApi.ADIAPI_OSDEgSendKeyComp(osdProgressbar1, 0x28);

}
```

2.3.6 ADIAPI_OSDEgSendKeyPage

Sends a key to any page without user intervention or gives focus to the page.

Method declaration:

```
publicvoidADIAPI_OSDEgSendKeyPage(IPagemyPage,BytekeyCode);
```

Parameters:

myPage:Any page which can receive a key input.

***keyCode:**keyCode to be sent.

Code window usage example:

```
//sending Up key to Page2

OsdApi.ADIAPI_OSDEgSendKeyPage(PageManager.Page2, 0x26);
```

2.3.7 ADIAPI_OSDEgIsAnimationRunning

Determines if a component is running an animation

Method declaration:

```
publicboolADIAPI_OSDEgIsAnimationRunning(HOBJ hObj);
```

Parameters:

hObj:OSD control name

Code window usage example:

```
//checking osdImage1 component is running an animation

boolret = OsdApi.ADIAPI_OSDEgIsAnimationRunning(osdImage1);
```

2.3.8 ADIAPI_OSDEgForceUpdate

Force OSD to update region data.

Method declaration:

```
publicvoidADIAPI_OSDEgForceUpdate(void);
```

Code window usage example:

```
OsdApi.ADIAPI_OSDEgForceUpdate();
```

2.4 Pressed Key Event Flow between Pages and Components

Some OSD components, implement a *RemoteKeyPress* event: *OSDListbox*, *OSDProgressBar*, *OSDMenubar*, *OSDKeyboard*, *OSDSlider*, *OSDIconListbox*, *OSDMulticolumnListbox*,*OSDTextbox*and*OSDIPTextBox*. This event is triggered when the user presses and releases 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 it is 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 keyflow for *OSDListbox*, *OSDProgressBar*, *OSDMenubar*, *OSDKeyboard*, *OSDSlider*, *OSDIconListbox*, *OSDMulticolumnListbox*,*OSDTextbox*and*OSDIPTextBox*components can be seen in Figure 3.

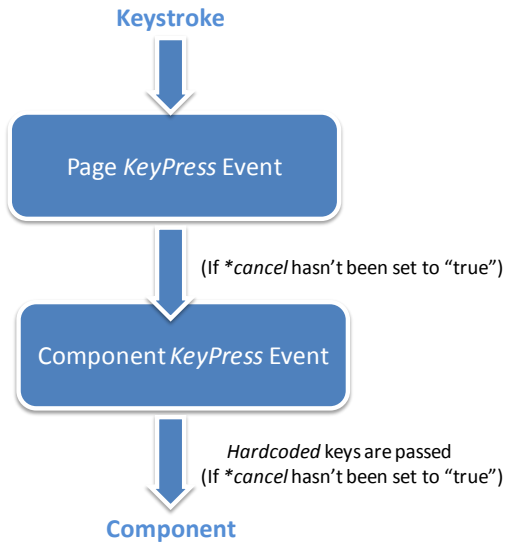


Figure 3: Pressed Key Event Flow

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 these properties, which are listed in Table 4. 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 4: Common Properties and Methods

Global Properties	Description
Name	Indicates the name used in the code to identify the 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.
zLayer	Sets the layer overlay priority of the component.
Size	Sets the size of the component in pixels.
Alpha	Sets the transparency of the component.
suspendLayout()	Allows several properties to be updated without redrawing component each time.
resumeLayout()	After properties have been updated, the component is released and can be drawn.

Note that *Size* can only be assigned through the *Property Navigator* of Blimp GUI, that is, itcannot be accessed or modified through the user code. This means that once the size of the components isset, It is fixed and cannot be modifiedin real time.

Each property is now described in detail. It is shown how each property is declared within the Blimp tool, and an example is given on how to use it in the code window. *suspendLayout()* and *resumeLayout()* methods are presented together.

Note that if the name of the component is changed in designer canvas, the name is updated automatically in code window when it is used. The undo/redo option is applicable to both designer canvas and code window.

3.1.1 Name

Sets the name of the object used in the code to identify the object. This is usually set through the GUI and cannot be changed in runtime.

1. When changing the name, a confirmation window should ask if names should be changed in code window as well.
2. Changing the name will also change the name used in code window.
3..Undo/Redo also supported for name change.

Property declaration:

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

3.1.2 Visible

Gets or sets the visibility status of the component.

Property declaration:

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

Code window usage example:

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

3.1.3 LocationX

Gets or sets the horizontal position of the component.

Property declaration:

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

Code window usage example:

```
osdListbox2.LocationX = 200;
```

3.1.4 LocationY

Gets or sets the vertical position of the component.

Property declaration:

```
public shortLocationY {get; set;}
```

Code window usage example:

```
osdListbox2.LocationY = 300;
```

3.1.5 Alpha

Gets or sets the transperancy of the component.

Property declaration:

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

Code window usage example:

```
osdLabel1.Alpha = 10;
```

3.1.6 zLayer

Gets or sets the priorityof the component. In the current version of the tool, sixteen different overlay priorities can be selected. Figure 4 shows an example of the use of *zLayer* property within a complex OSD design.

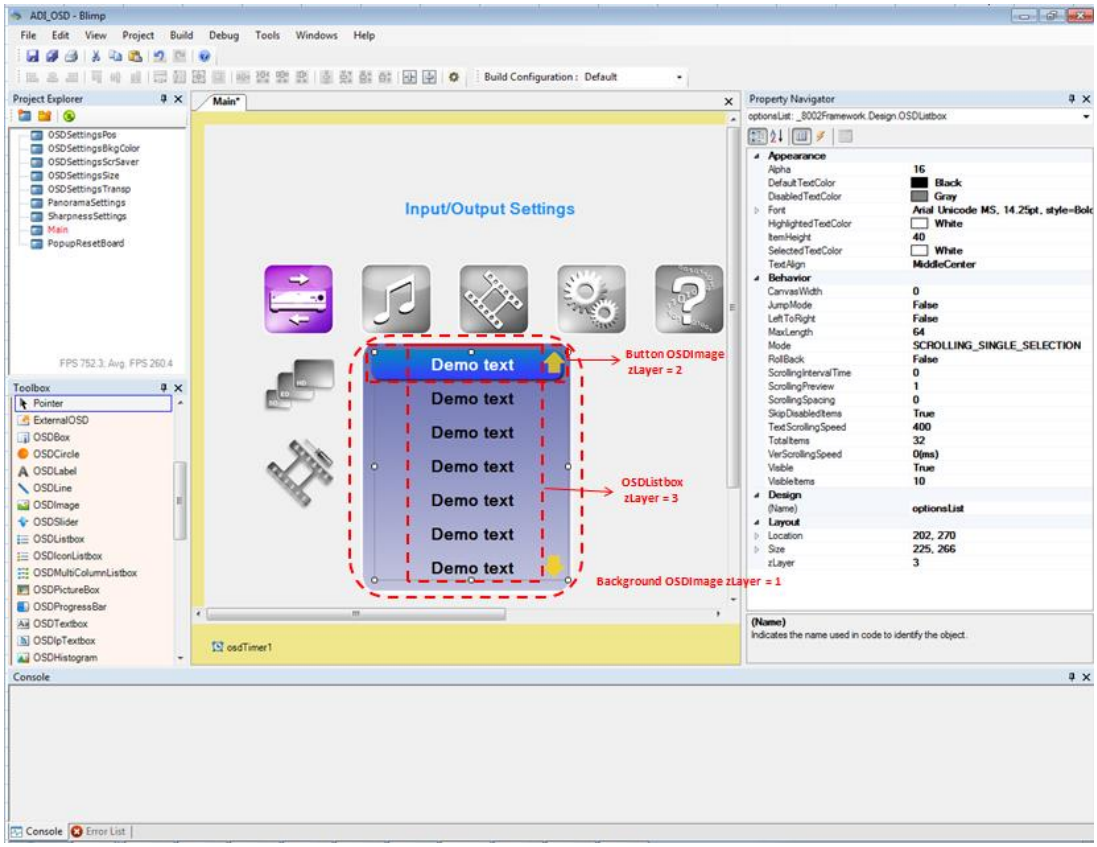


Figure 4: Example Usage of zLayer Property

Property declaration:

```
public bytezLayer {get; set;}
```

Code window usage example:

```
osdLabel1.zLayer = 10;
```

3.1.7 SuspendLayout and ResumeLayout

When loading a page when many components have to be initialized, or in pages with many components changing several of its properties at a fast pace, many writes to the components' properties or methods need to be done. Each of these accesses means that the component needs to be redrawn each time, causing SPI utilization to increase dramatically, thus consuming more CPU time and holding the system for a longer time.

A simple example is as follow for the histogram component:

```
public void Load()
{
    byte i;

    osdHistogram1.TotalBins = 30;
    osdHistogram1.MaxValue = 30;
    osdHistogram1.MinValue = 0;
    osdHistogram1.Spacing = 1;

    for (i = 0; i < 30; i++)
    {
        osdHistogram1.setBinValue(i, (short)i);
    }
}
```

In this example, once the Blimp code is compiled, a call is made to redraw the component each time a property is changed. In order to avoid this issue, all the components implement two methods, called *suspendLayout()* and *resumeLayout()*. *suspendLayout()*holds the drawing of the component on the screen until the user resumeLayout is called after the user has finished updating the component properties. Then, the call to *resumeLayout()*would let the hardware draw the updated component.

A simple example on how suspendLayout and resumeLayout functions are to be used is as follow:

```
public void Load()
{
    byte i;

    osdHistogram1.suspendLayout();

    osdHistogram1.TotalBins = 30;
```

```
        osdHistogram1.MaxValue = 30;
        osdHistogram1.MinValue = 0;
        osdHistogram1.Spacing = 1;

        for (i = 0; i < 30; i++)
        {
            osdHistogram1.setBinValue(i, (short)i);
        }

        osdHistogram1.resumeLayout();
    }
```

It is recommended to use these two methods even if updatingjust two properties; there will always be a performance benefit.

It is recommended to use suspendLayout, resumeLayout when changing more than one property of component in script page. So that we can reduce the SPI utilization and CPU time. It should be used if a component animation in running state to avoid animation corruptions.

3.2 Enumeration Types

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

Table 5: Enumeration 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
OSDListboxMode	Defines behavior of selected items within an*OSDlistbox* component
OSDAnimationEffectType	Defines behavior of*OSDImage, Page*animation effect
OSDAnimationEffectDirection	Defines direction of *OSDImage, Page*animation effect

3.2.1 OSDContentAlignment

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: Content is 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

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

Orientates vertically or horizontally the component. Can be used on the*OSDProgressbar*, *OSDMenubar* and *OSDHistogram* components.This is usually set through the GUI although it may also be changed atrun time.

Property declaration:

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

VERTICAL: Vertical orientation

HORIZONTAL: Horizontal orientation

Code window usage example:

```
//For example, assuming that osdMainMenu is a vertically-orientated
OSDMenubar component, its orientation //can be changed to horizontally-
orientated during runtime through
osdMainMenu.Orientation = OSD_ORIENTATION.HORIZONTAL;
```

3.2.4 OSDListboxMode

Used to define the behavior of the selected items within an *OSDlistbox* component. This is usually set through the GUI only and it cannot be changed in runtime.

Property declaration:

```
public enum {
SCROLLING_SINGLE_SELECTION,
SCROLLING_SINGLE_SELECTION_ITEM_FIRST,
SCROLLING_MULTI_SELECTION,
SCROLLING_NON_SELECTION
} OSD_LISTBOXMODE;
```

SCROLLING_SINGLE_SELECTION: Only one selected item per listbox.

SCROLLING_SELECTION_FIRST: Highlighteditem will always stay in the first position of the listbox.

SCROLLING_MULTI_SELECTION: Several selections are allowed within the same listbox.

SCROLLING_NON_SELECTION: User can move through the *listbox* but no item can be selected.

Code window usage example:

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

//Change the listbox operation mode to multi selection
VSPListbox.ItemText[0] = "MNR";
VSPListbox.ItemText[1] = "BNR";
VSPListbox.ItemText[2] = "Sharpness Filter";

OsdApi.ADIAPI_OSDEgSetFocusComponent(VSPListbox);
}
```

3.2.5 OSDAnimationEffectType

Used to define theanimation effect behavior of *OSDImage component and Page*. This is usually set through the GUI although it may also be changed at runtime.

Property declaration:

```
publicenumANIMATIONEFFECTTYPE
  {
None,
```

```
    FlyIn,
    FlyOut,
    FadeIn,
    FadeOut
}
```

FlyIn: A fly-in effect shall make the component appear from desired direction to specified position on OSD. Supported direction will be top, bottom, left, right.

FlyOut: A fly-out effect shall make the component appear from desired direction to specified position on OSD.Supported direction will be top, bottom, left, right.

FadeIn: A Fade in effect shall cause the component to go from alpha zero to full alpha when visibility is set. And the fade effect shall start when visibility goes to true and will evenly fade in time specified by speed

FadeOut: A Fade out effect shall cause the component to go from full alpha to zero alpha when visibility is set to False. And the fade effect shall start when visibility goes to true and will evenly fade in time specified by speed

Code window usage example:

```
//Changing the OsdImage animation effect as FadeIn

osdImage1.AniEffectType = OSD_ANIEFFECTTYPE.FADEIN;

//Changing the Page animation effect as FadeOut

PageManager.Page1.AniEffectType = OSD_ANIEFFECTTYPE.FADEOUT;
```

3.2.6 AnimationEffectDirection

Used to define theanimation effect dirction of *OSDImage component and Page*. This is usually set through the GUI although it may also be changed at runtime.

Property declaration:

```
publicenumANIMATIONDIRECTION
 {
    Left,
    Right,
    Top,
    Bottom,
 }
```

Code window usage example:

```
//Changing the OsdImage animation effect direction as bottom

osdImage1.AniEffectDirection = OSD_ANIEFFECTDIRECTION.BOTTOM;
```

```
//Changing the Page animation effect direction as Top

PageManager.Page1.AniEffectDirection = OSD_ANIEFFECTDIRECTION.TOP;
```

3.2.7 Language Enumeration Types

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_resource.h* file, within the *Release* folder, and should not be modified by the user.

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

3.3 Pages and Components

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 Properties

Every OSD Page implements these properties, which are listed inTable 6.

Table 6: Page Properties

Global Properties	Description

Static	Disables page buffering for this page and always keep loaded in memory.
Location (X,Y)	Sets thePage location of X and Y
zLayer	Sets the layer overlay priority of the component.
CanvasColor	Background color of the Blimp designer canvas. Only used in blimp designer view.

3.3.1.1 Static

Disables page buffering for this page and always keep loaded in memory. Please refer section 18 for more information.

3.3.1.2 Location (X, Y)

Sets the page location of X and Y. Page location can be changed in run time by SetPageLocationXY method.

Code window usage example:

```
PageManager.Page1.SetPageLocationXY(50,50);
```

3.3.1.3 zLayer

Sets the layer overlay priority of the component.

Note: Page zLayer can be changed in run time using the OSDEG_setPageZlayer API.

3.3.2 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 7: Pages Events

Page Event	Short Description
OnFocus	Occurs when page gains main active focus
OnUnFocus	Occurs when page loses main active focus
Load	Occurs when page has finished loading
AnimationEnd	Occurs when the page animation finished
AnimationStart	Occurs when the page animation start
RemoteKeyPress	Occurs when page has focus and user presses and releases a key
OnShow	Occurs when the page is shown
OnHide	Occurs when the page is hidden

3.3.2.1 OnFocus

Occurs when the page gains the main active focus.

Syntax: `privatevoidOnFocus(void)`

Code window usage example:

```
private void Page1_OnFocus()
{
    osdLabel1.Text = "Activate page event triggered";
}
```

3.3.2.2 OnUnFocus

Occurs when the page loses the main active focus.

Syntax: `privatevoidOnUnFocus(void)`

Code window usage example:

```
private void Page1_OnUnFocus()
{
    osdLabel1.Text = "De-activate page event triggered";
}
```

3.3.2.3 Load

Occurs when the page has finished loading. At the current version of the tool, the ancillary Load() method is still available. This is done for backwards compatibility reasons and it will be removed in future revisions of the tool. At the moment, the ancillary Load() method is executed before the Page Load() event. It is recommended to use only the Page Load event.

Syntax: `privatevoidLoad(void)`

Code window usage example:

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

3.3.2.4 AnimationStart

Occurs when the page animation start

Syntax: `privatevoidAnimationStart(void)`

Code window usage example:

```
private void Page1_AnimationStart()
{
osdLabel1.Text = "Animation start event triggered";
}
```

3.3.2.5 AnimationEnd

Occurs when the page animation finished

Syntax: `privatevoidAnimationEnd(void)`

Code window usage example:

```
private void Page1_AnimationEnd()
{
  osdLabel1.Text = "Animation end event triggered";
}
```

3.3.2.6 RemoteKeyPress

Occurs when the page has focus and the user presses and releases a key. See the Pressed Key Event Flow between Pages and Componentssection on page 21 for more information.

Syntax: `privatevoidRemoteKeyPress(Byte* keyCode, Boolean* cancel)`

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,a 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, 100);
        OsdApi.ADIAPI_OSDEgHidePage(PageManager.Page1, 0);

        OsdApi.ADIAPI_OSDEgSetFocusPage(PageManager.Page2);
    }

}
```

3.3.2.7 OnShow

Occurs when the page is shown.

Syntax: `privatevoidOnShow(void)`

Code window usage example:

```
private void Page1_OnShow()
{
    osdLabel1.Text = "The page is shown";
}
```

3.3.2.8 OnHide

Occurs when the page is hidden.

Syntax: `privatevoidOnHide(void)`

Code window usage example:

```
private void Page1_OnHide()
{
    osdLabel1.Text = "The page is hidden";
}
```

3.3.3 OSDLabel

OSDLabel components are used to display text. The user can also write code that changes the displayed text by a label in response to events at run time. This component cannot be the target of the page focus, and it has one associated event.

Table 8: OSDLabel Properties

Property or Method	Short Description
Text	Sets the text to be displayed.
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[].
FgColor	Gets or sets foreground color of text.
Orientation	Sets the direction of the text.
TextAlign	Gets or sets alignment of the text.
Font	Gets or sets the font.
LineSpacing	Sets Line spacing for multiline mode in pixels from bottom to bottom of each line.
setTextFormat()	Sets a formatted text string in ascii. This method is very similar to the sprint function in C.
setTextFormatW()	Sets a formatted text string in unicode. This method is very similar to the sprint function in C.
ConstText	Sets a string from the multilanguage string table. The StringID can be used to specify the string using StringManager.
LeftToRight	Gets or sets direction of scrolling.
ScrollingEnabled	Gets or sets if scrolling functionality is enabled.
ScrollingBounceBack	Causes the direction of the scrolling text to alternate direction when text reaches the end of the display area.
ScrollOffDisplay	Text will start scrolling from outside of display area and end scrolling when text is completely outside display area.
ScrollTextRepeat	Causes the text to concatenate after the scrolling spacing
MultilineEnabled	Sets if the component can display multiple lines of text.
TextScrollingSpeed	Gets or sets the speed of the scrolling (pixels/seconds).
ScrollingIntervalTime	Gets or sets amount of time control will wait until scrolling is restarted.
ScrollingSpacing	Gets or sets the spacing between the text during scrolling mode. (pixels).
MaxLength	Sets maximum number of characters *OSDLabel* can contain at any time.
CanvasHeight	Sets the height of DDR2 display canvas area for multiline text. Used only if multiline is enabled. 0 = Auto.
CanvasWidth	Sets the width of DDR2 display canvas area for scrolling text. Used only if scrolling is enabled. 0 = Auto.

AutoHeight	Enables automatic control height based on font size for single line text. Note that this is not valid for multiline text or font style configuration used.

Table 9: OSDLabel Events

Event	Short Description
ScrollingFinish	Triggered when text scrolling has finished.

3.3.3.1 Text

Sets the text to be displayed. The current version will support the Unicode characters.While passing the Unicode characters, those are converted to UTF-8 format and the firmware will encode the Unicode characters. Below describes the range and the format of these different octet types. The letter x indicates bits available for encoding bits of the character number.

- 00000000 -- 0000007F: 0xxxxxxx
- 00000080 -- 000007FF: 110xxxxx 10xxxxxx
- 00000800 -- 0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx

Note: In Text, ItemText property, the given Unicode characters should lie in any one of the above range. The mixed up Unicode chars won't be supported in blimp.

Property declaration:

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

Code window usage example:

```
//The following example shows how it is possible to assign the items of
an OSDListbox and the Text

//property of two OSDLabel when the OSD state gets changed.

case MAIN_MENU_STATES.INPUT_SELECTOR:
    mainListBox.TotalItems = 6;
    mainListBox.VisibleItems = 6;
    mainListBox.ItemText[0] = "HDMI 1";
    mainListBox.ItemText[1] = "HDMI 2";
    mainListBox.ItemText[2] = "HDMI 3";
    mainListBox.ItemText[3] = "CVBS 1";
    mainListBox.ItemText[4] = "CVBS 2";
```

```
        mainListBox.ItemText[5] = "Components 1";
        descriptionLabel.Text = "Select the input connector.";
        titleLabel.Text = "INPUT SELECTOR";
        break;
```

3.3.3.2 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;

}
```

3.3.3.3 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;
}
```

3.3.3.4 FgColor

Gets or sets the foreground color of the text.

Property declaration:

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

Code window usage example:

```
// 32 bits value {A,R,G,B}
//Sets the color to full opaque black
osdLabel1.FgColor = 0xFFFFFFFFu;
```

3.3.3.5 Orientation

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;}
```

3.3.3.6 TextAlign

This property is used 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, BOTTOMRIGHT) and the text will be aligned with the right edge of the labels to align with your controls.

Property declaration:

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

Code window usage example:

```
//To align the text to the top right corner
osdLabel1.TextAlign = OSD_CONTENTALIGNMENT.TOPRIGHT;
```

3.3.3.7 Font

Gets or sets the font being used on the *OSDLabel* component.

For more details please refer the sectionSelecting fonts.

3.3.3.8 LineSpacing

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

Property declaration:

```
public UINT8LineSpacing { set;}
```

3.3.3.9 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 10: 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");
```

3.3.3.10 setTextFormatW

Sets a formatted text string in unicode. This method is very similar to the sprint function in C.

Method declaration:

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

Parameters:

format: String to be formatted and stored. Ordinary characters (not limited to ASCII characters, any Unicode character can be used) 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 11: List of supported formats in setTextFormatW

%c	a character with the given number
%s	a string
%d	a signed integer, in decimal
%x	an unsigned integer, in hexadecimal

Code window usage example:

```
//For this example, assume that osdLabel1 reads "Hello World" and
osdLabel2 is initialized
private void labelPage_Load()
{
  //Create a 16-Bit String to hold a Unicode Value
  ushort []myUnicodeString = new ushort [12];

  //Copy contents of OsdLabel1 into the string
  osdLabel1.readTextW(myUnicodeString,12);

  //osdLabel2 reads now "Hello World".
// Note that using setTextFormat here is not possible since the
//string contains 16-Bit Unicode chars instead of 8-Bit ASCII chars.
  osdLabel2.setTextFormatW("The Unicode string is: %s",myUnicodeString);
}
```

3.3.3.11 ConstText

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.

Method declaration:

```
public UINT32[] ConstText
```

Code window usage example:

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

StringID will always be uppercase even if defined as lowercase.

3.3.3.12 LeftToRight

Gets or sets the direction of scrolling. If set to true, the text scrolls from left to right; if set to false, scrolls from right to left.

Method declaration:

```
publicboolLeftToRight{get; set;}
```

Code window usage example:

```
osdLabel1.LeftToRight = true;
```

3.3.3.13 ScrollingEnabled

Gets or sets if the scrolling functionality (vertical or horizontal) is enabled. If *ScrollingEnabled* is set to '1' and *Multiline* is also set, then the scrolling is vertical; if *Multiline* is set to '0', it is horizontal (it is not possible to have both horizontal and vertical scrolling enabled at the same time). Setting *ScrollingEnabled* to '0' disables completely any scrolling.

Method declaration:

```
publicboolScrollingEnabled{get; set;}
```

Code window usage example:

```
osdLabel1.ScrollingEnabled = true;
```

3.3.3.14 ScrollingBounceBack

Causes the direction of the scrolling text to alternate direction when text reaches the end of the display area. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicbool ScrollingBounceBack{get;set;}
```

3.3.3.15 ScrollOffDisplay

Text will start scrolling from outside of display area and end scrolling when text is completely outside display area.This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT8 ScrollOffDisplay{get;set;}
```

Note that ScrollingBounceBack and ScrollTextRepeat should be set to false to view the ScrollOffDisplay behavior

3.3.3.16 ScrollTextRepeat

Causes the text to concatenate after the scrolling spacing .It is usually set in the GUI although it may also be changed during runtime.

Method declaration:

```
publicbool ScrollTextRepeat{get;set;}
```

Code window usage example:

```
osdLabel1.ScrollTextRepeat = true;
```

Note that ScrollingBounceBack should be set to false to view the ScrollTextRepeat behavior

3.3.3.17 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:

```
publicboolMultilineEnabled{get; set;}
```

Code window usage example:

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

3.3.3.18 TextScrollingSpeed

Gets or sets the speed of the scrolling. The speed is defined in pixels/sec. Range[0 - 2000]

Method declaration:

```
publicUINT8 TextScrollingSpeed{get; set;}
```

Code window usage example:

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

3.3.3.19 ScrollingIntervalTime

Gets or sets the amount of time the control waits until scrolling is restarted.The ScollingIntervalTime is defined in frames.This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 ScrollingIntervalTime{get; set;}
```

3.3.3.20 ScrollingSpacing

Gets or sets the spacing between the text during scrolling mode. The scrollingspacing is defined in pixels. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 ScrollingSpacing{get; set;}
```

3.3.3.21 MaxLength

Sets the maximum amount of Unicode characters the *OSDLabel* can contain at any time. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 MaxLength{get; set;}
```

3.3.3.22 CanvasHeight

Sets the canvas area of the *OSDLabel*. This property must be set if vertical scrolling is going to be used. *CanvasHeight* must be big enough to fit the whole text chunk or the animation will jump. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 CanvasHeight{get; set;}
```

3.3.3.23 CanvasWidth

Sets the canvas area of the *OSDLabel*. Used for horizontal scrolling. This property can be set if horizontal scrolling is going to be used. *CanvasWidth* must be big enough to fit the whole text chunk or the animation will jump. This is usually set once on the GUI and not changed during run time. The value 0 means Auto.It would only be used if user wants to reserve a area in DDR2 memory and change the text size dynamically without causing reallocation of DDR2 memory.

Method declaration:

```
publicUINT16 CanvasWidth{get; set;}
```

3.3.3.24 AutoHeight

Enables automatic control height based on font size for single line text. Note that this is not valid for multiline text or font style configuration used. This is usually set once on the GUI and not changed during run time.

AutoHeight can also be set through the canvas window as shown below:

3.3.3.25 ScrollingFinish

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*.

Syntax: `privatevoidScrollingFinish(void)`

3.3.4 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* developsscrolling functionality. The color of the text for this component changes depending on the state of the item (default, highlighted, selected and disabled). Thisis set from the properties *DefaultTextColor*, *HighlightedTextColor*, *SelectedTextColor*, *DisabledTextColor*.

This component may be the target of the page focus, which is needed 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 inTable 13.

Table 12: OSDListbox Properties

Property or Method	Short Description
TotalItems	Gets or sets the maximum number of items that the list can contain.
VisibleItems	Gets or sets the number of items that list will display.
SelectedIndex	Gets or sets current selected item index.
Font	Gets or sets the font.
TextAlign	Gets or sets the alignment of the text.
DefaultTextColor	Gets or sets the default color for the list items text.
HighlightedTextColor	Gets or sets for the color for the highlighted item in the list.
SelectedTextColor	Gets or sets for the color for the selected item(s) in the list.
ItemText	Sets an item text as a null terminated ascii string from a buffer of type char[].
ItemTextW	Sets an item text as a unicode string from a buffer of type uint16[].
readItemTextW()	Gets text of any *OSDListbox* item and stores it in a buffer
ItemConstText	Gets or sets an text as a string from the multilanguage string table. The StringID can be used to specify the string using StringManager.
Mode	Gets or sets the scrolling and selection behavior of the listbox.
ItemHeight	Gets or sets the height of each item in the list box in pixels.
ItemStatus	Gets or sets checked property of an item.
FocusIndex	Gets or sets current highlighted item index.
setItemTextFormat()	Sets an item text as a formatted text string in ascii. This method is very similar to the sprint function in C.
DisabledItem	Gets or sets items that appear disabled in the list.
DisabledTextColor	Gets or sets for the color for disabled items in the list.
HiddenItems	Gets or sets hide property of an item.
OffsetIndex	Gets or sets offset of the *OSDListbox.* It takes hidden items into account.
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.

TextScrollingSpeed	Gets or sets horizontal speed of text scrolling animation.
JumpMode	Sets whether the list will go back to the first item on the list once the last one has been reached and vice versa.(without continuous scrolling).
SkipDisabledItems	If True, selection will jump over disabled items when scrolling through listbox items.
LeftToRight	Gets or sets the direction of the scrolling.
ScrollingIntervalTime	Gets or sets the amount of time the control will wait until scrolling is restarted.
ScrollingSpacing	Gets or sets the spacing between the text during scrolling mode. (pixels)
CanvasWidth	Gets or sets canvas area of *OSDListbox*. Used for horizontal scrolling.
ScrollingPreview	Gets or Sets the number of items which will be visible above or below the highlighted item in the list to allow rpeview of the next item.
VerScrollingSpeed	Range[50-5000mS].Sets the speed of the vertical scrolling animation through items.0 to disable.
MaxLength	Sets maximum number of Unicode characters that any of the items can contain at any time

Table 13: Hardcoded 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

Table 14: OSDListbox 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 has been completed
VertScrollingFinish	Occurs when the vertical scrolling between list items has been completed

3.3.4.1 TotalItems

Gets or sets the 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;
```

3.3.4.2 VisibleItems

Gets or sets the number of items that list will display, which must be less than or equal *toTotalItems*. If *VisibleItems*<TotalItems, the listbox can be scrolled..

Property declaration:

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

Code window usage example:

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

3.3.4.3 SelectedIndex

Gets or sets current selected item index. Note that this property can only be checked when *Mode* is set to SCROLLING_SINGLE_SELECTION.

Property declaration:

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

Code window usage example:

```
//Initialize osdListbox1
    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)
    {
//We get the same result with both instructions. Note however that, if
using SCROLLING_MULTI_SELECTION, //the second implementation would still
work: "index" value would be the last selected item.
        osdLabel1.setTextFormat("HDMI Input Selected %d",
osdListbox1.SelectedIndex + 1);
        osdLabel2.setTextFormat("HDMI Input Selected %d", index + 1);
    }
```

3.3.4.4 Font

Gets or sets the font.

For more details please refer the section Selecting fonts.

Code window usage example:

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

//It is also possible to assign fonts between two components
osdListbox1.Font = osdListbox2.Font;
```

3.3.4.5 TextAlign

Gets or sets the alignment of the text to match the layout of controls on the page. 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;}
```

Code window usage example:

```
//Align all the items within the OSDListbox to the middle left
osdLabel1.TextAlign = OSD_CONTENTALIGNMENT.MIDDLELEFT;
```

3.3.4.6 DefaultTextColor

Gets or sets the default color for the list items text. This is usually set through the GUI although it may be changed atrun time.

Property declaration:

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

Code window usage example:

```
//Set default text to Green color
osdListbox1.DefaultTextColor = 0xFF00FF00u;
```

3.3.4.7 HighlightedTextColor

Gets or 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:

```
//Set default text to Yellow color
osdListbox1.HighlightedTextColor = 0xFFFFFF00u;
```

3.3.4.8 FocusIndex

Gets or sets the current highlighted item index. Note that by default, the highlighted item in a list will always be the first element.

Property declaration:

```
public byte UINT32 FocusIndex{get; set;}
```

Code window usage example:

```
//We can change the behavior of the first item being highlighted by
default
    public void Load()
    {
        osdListbox1.ItemText[0] = "HDMI 1";
        osdListbox1.ItemText[1] = "HDMI 2";
        osdListbox1.ItemText[2] = "HDMI 3";
        osdListbox1.ItemText[3] = "HDMI 4";

        osdListbox1.FocusIndex = 2;

        OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
    }
```

```
//These two instructions will return the same string
private void osdListbox1_HighlightedItemChanged(Byte index)
    {
        osdLabel1.setTextFormat("The Highlighted HDMI Input is %d",
osdListbox1.FocusIndex + 1);
        osdLabel2.setTextFormat("The Highlighted HDMI Input is %d",
index + 1);
    }
```

3.3.4.9 SelectedTextColor

Gets or sets for 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.

Note that when the listbox item is selected and focused, then it will be displayed using color as defined by SelectedTextColor

Property declaration:

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

Code window usage example:

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

3.3.4.10 ItemText

Sets an item text as a null terminated ascii string from a buffer of type char[]. To read the text set in an item, use *readItemTextW*() method.

Property declaration:

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

Code window usage example:

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

3.3.4.11 ItemTextW

Sets an item text as a unicode string from a buffer of type uint16[].

Code window usage example:

```
private void ReadBufferString()
{
```

```
    ushort[] buffer = new ushort[15];

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);

    osdListbox1.TotalItems = 3;
    osdListbox1.ItemText[0] = "First Item";
    osdListbox1.ItemText[1] = "Second Item";
    osdListbox1.ItemText[2] = "Third Item";

    //Read 11 characters from osdListbox1 and put the read string into
buffer
    osdListbox1.readItemTextW(0,buffer,11);

    //osdListbox1.ItemTextW[1] reads now "First Item"
    osdListbox1.ItemTextW[1] = buffer;
}
```

3.3.4.12 readItemTextW

Gets the text of any *OSDListbox* item and stores it into a buffer. The read text can be any Unicode character.

Method declaration:

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

buffer: Array where the read characters are going to be stored in. It should be long enough to hold the value set in *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) which are going to be read from the item. For example, if the item string length is 11, size should be set to 12.

Code window usage example:

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

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);

    osdListbox1.TotalItems = 3;
    osdListbox1.ItemText[0] = "First Item";
    osdListbox1.ItemText[1] = "Second Item";
```

```
    osdListbox1.ItemText[2] = "Third Item";

    //Read 11 characters from osdListbox1 and put the read string into
buffer
    osdListbox1.readItemTextW(0,buffer,11);

    //osdListbox1.ItemTextW[1] reads now "First Item"
    osdListbox1.ItemTextW[1] = buffer;
}
```

3.3.4.13 ItemConsText

Gets or sets an 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 *LanguageSettings* menu in Blimp.

Method declaration:

```
public UINT32[] ItemConstText
```

Code window usage example:

```
osdListbox1.ItemConstText[1] = StringManager.AUDIO_SETTINGS;

//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
```

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

3.3.4.14 Mode

Gets or sets the scrolling and selection behavior of the listbox. This is usually set once on the GUI and not changed during run time.

Property declaration:

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

3.3.4.15 ItemHeight

Gets or 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.

Property declaration:

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

Code window usage example:

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

3.3.4.16 IsScrolling

Gets the highlighted item has horizontal scrolling. Returns TRUE is the highlighted item has horizontal scrolling enabled. FALSE otherwise.

Property declaration:

```
public boolIsScrolling{get; set;}
```

Code window usage example:

```
boolScrolling_flag = osdListbox1.IsScrolling;
```

3.3.4.17 setItemTextFormat

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 10 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);
            }
        }
```

3.3.4.18 DisabledItems

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:

```
publicbool[] DisabledItems{get; set;}
```

Code window usage example:

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

3.3.4.19 DisabledTextColor

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;
```

3.3.4.20 HiddenItems

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;
```

3.3.4.21 OffsetIndex

Gets or sets 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 also takes into account hidden items (see the *HighlightedSlot* property for more information).

This propriety is useful when using some kind of selection box around the items on the *OSDListbox* and need to position it correctly as the user scrolls up and down the list, as shown in Figure 5.

Property declaration:

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

Code window usage example:

```
//The position of a selection border can be properly determined by using
the offset
listBox1SelBorder.LocationY = ( yOrigin + (OffsetIndex * yStep ));
```

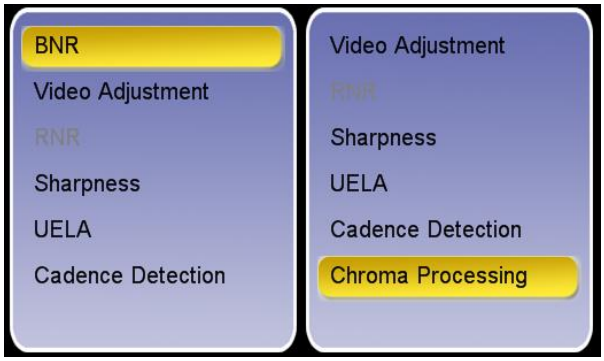


Figure 5: Offset Index

:

OffsetIndex is 0 in the left list and it matches FocusIndex as long as vertical scrolling has not happened. It is5 in the right list.

3.3.4.22 HighlightedSlot

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 : _8002Emulator.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);
    }
}
```

Figure 6: Highlighted slot with no hidden Items

There are no hidden items in this list. Note how HighlightedSlot property reads the same as OffsetIndex

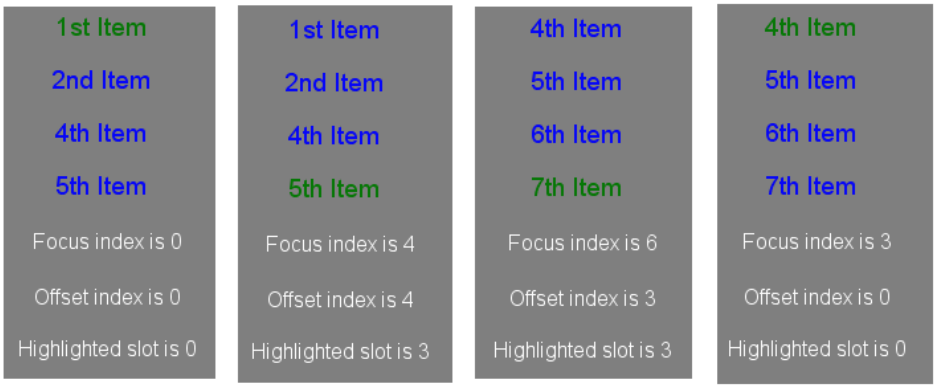


Figure 7: Highlighted slot with one Hiden Items

There is one hidden item in this list. Note how HighlightedSlot property does not read the same as the OffsetIndex property.

3.3.4.23 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. This is usually set once on the GUI and not changed during run time.

Property declaration:

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

3.3.4.24 TextScrollingSpeed

Gets or sets the horizontal speed of the text scrolling animation. A value of '0' means horizontal scrolling disabled. The speed is defined in pixels/ms.

Method declaration:

```
publicUINT8 TextScrollingSpeed{get; set;}
```

Code window usage example:

```
osdListbox1.HorScrollingSpeed = 20;
```

3.3.4.25 JumpMode

Sets whether the list will go back to the first item on the list once the last one has been reached and vice versa(without continuous scrolling). This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicbool JumpMode{get; set;}
```

3.3.4.26 SkipDisabledItems

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:

```
publicbool SkipDisabledItems{get; set;}
```

3.3.4.27 LeftToRight

Gets or sets the direction of the scrolling. If set to true, the text scrolls from left to right; if set to false, scrolls from right to left.

Method declaration:

```
publicboolLeftToRight{get; set;}
```

Code window usage example:

```
osdListbox1.LeftToRight = true;
```

3.3.4.28 ScrollingIntervalTime

Gets or sets the amount of time the control waits until horizontal text scrolling is restarted. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 ScrollingIntervalTime{get; set;}
```

3.3.4.29 ScrollingSpacing

Gets or sets the spacing between the text during scrolling mode (pixels). This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 ScrollingSpacing{get; set;}
```

3.3.4.30 CanvasWidth

Sets the canvas area of the *OSDListbox*. Used for horizontal scrolling. This property must be set if horizontal scrolling is going to be used. *CanvasWidth* must be big enough to fit the whole text chunk or the animation will jump. It is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 CanvasWidth{get; set;}
```

3.3.4.31 ScrollingPreview

Gets or Sets the number of items which will be visible above or below the highlighted item in the list to allow rpeview of the next item. Figure 8 and Figure 9 show how an *OSDListBox* would look when scrolling through its items, with PreviewItem = 0 and PreviewItem = 1, respectively. This is usually set once on the GUI and not changed during run time.

Property declaration:

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

Code window usage example:

```
//Code used in listboxes shown in below figures and set ScrollingPreview
as 1.
private void Page1_Load()
{
    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox);

    osdListbox.TotalItems = 10;
    osdListbox.VisibleItems = 5; //Scrolling Enabled

    osdListbox.ItemText[0] = "BNR";
    osdListbox.ItemText[1] = "MNR";
    osdListbox.ItemText[2] = "RNR";
    osdListbox.ItemText[3] = "Sharpness";
    osdListbox.ItemText[4] = "ULAI";
    osdListbox.ItemText[5] = "Cadence";
    osdListbox.ItemText[6] = "Chroma Processing";
    osdListbox.ItemText[7] = "Cue Correction";
    osdListbox.ItemText[8] = "Panorama";
    osdListbox.ItemText[9] = "Game Mode";
}
```

Figure 8: HigPreviewItem = 0

The focused item is always the last within the visible items, and scrolling triggers from this position when moving to the next item.

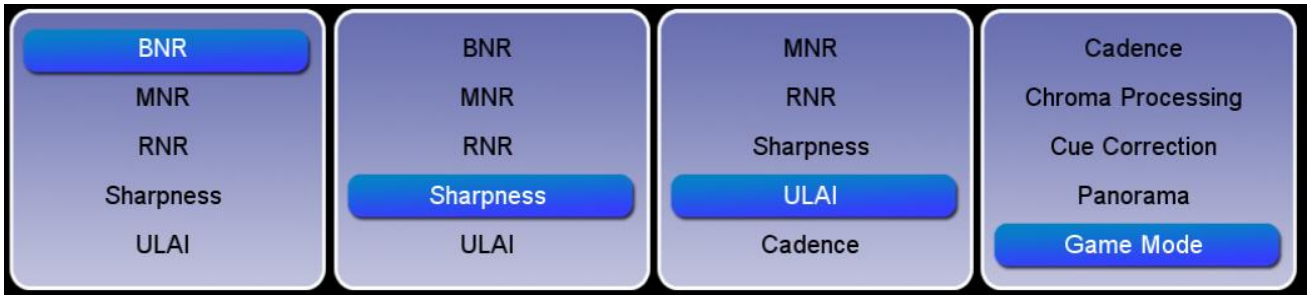


Figure 9: PreviewItem = 1

Note how, when scrolling the list, the focused item is always the second last, unless the user gets to the actual last item of the list.

3.3.4.32 VerScrollingSpeed

Gets or sets the speed of the vertical scrolling animation. A value of '0' means vertical scrolling disabled. The speed is defined in ms.
Option list as below can be selected:
Very fast - 250
Fast - 500
Medium - 1000
Slow - 1500
Very slow - 2000
A custom value can also be entered directly. This should be combo box /text box combination with only number. Range should be 50 - 5000

Method declaration:

```
publicUINT32 VerScrollingSpeed{get; set;}
```

Code window usage example:

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

3.3.4.33 MaxLength

Sets the maximum number of Unicode characters that any of the items can contain at any time. This is usually set once on the GUI and not changed during run time.

Method declaration:

```
publicUINT16 MaxLength{get; set;}
```

3.3.4.34 HighlightedItemChanged

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

Syntax: `privatevoidHighlightedItemChanged (Byte index)`

Parameters:

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)
    {
        osdLabel1.setTextFormat("The Highlighted HDMI Input is %d",
index + 1);
    }
```

3.3.4.35 SelectedItemChanged

This event occurs when the selected item is changed, that is, when the user sends the spacebar key.

Syntax: `privatevoidSelectedItemChanged (Byte index, Boolean newStatus)`

Parameters:

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)
    {
//We get the same result with both instructions. Note however that, if
using SCROLLING_MULTI_SELECTION, //the second implementation would still
work: "index" value would be the last selected item.
        osdLabel1.setTextFormat("HDMI Input Selected %d",
osdListbox1.SelectedIndex + 1);
        osdLabel2.setTextFormat("HDMI Input Selected %d", index + 1);
    }
```

3.3.4.36 RemoteKeyPress

Syntax: `private void RemoteKeyPress (Byte *keyCode, Boolean *cancel)`

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

Parameters:

***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 OSDListboxwill 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 13.

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 tothe . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

Code window usage example:

```
//We could use right and left arrow keys to enable/disable other
submenus
private void osdListbox1_RemoteKeyPress(Byte *keyCode, Boolean *cancel)
{
    if (osdListbox1.FocusIndex == 0)
    {
       if (*keyCode == 39) //Right Arrow => Show Submenu
        {
           osdListbox1.Visible = False;
           osdListbox_Submenu.Visible = True;
 OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox_Submenu);
        }
}
}
```

3.3.4.37 ScrollingFinish

Syntax: `private voidScrollingFinish (void)`

This event occurs when the horizontal scrolling 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*.

3.3.4.38 VertScrollingFinish

Syntax: `private voidVertScrollingFinish (void)`

This event occurs when the vertical scrolling is completed.

3.3.5 OSDImage

It is used for containing an image. The*OSDImage*component can be used to display static images and animations (which are composed of tiles or frames). The process used to assign an image to the component is explained in Section 3.3.5.16. This component cannot receive focus, and has three associated event.

Table 15: OSDImage Properties

Property or Method	Short Description
ColorDepth	Sets the color format.
Scaling	Sets if image will scale to component size.
SizeMode	Controls how the osdImage will handle control sizing.
loadImageData	Dynamically loads an image to DDR2 memory.
ReadImageData	Reads raw Image data from DDR2 memory into MCU RAM.
AnimationEnabled	Gets or sets if animation will be active.

AnimationSpeed	Gets or sets the animation speed in frames per second.
AnimationType	Gets or Sets if the animation will loop continuously, bounce or run only once.
InitialFrame	Gets or sets the initial frame from the list of images when animation is enabled. (Starts at index 0).
FinalFrame	Gets or sets the final frame from the list of image when animation is enabled.
CurrentFrame	Gets or sets the starting frame to begin the animation with when animatino is enabled.
AnimationEffectDirection	A property shall specify the direction for the animation effect.
AnimationEffectType	The effect type property shall specify which animation effect is to be enabled.
AnimationEffectStartposition	A property shall specify the start position of animation effect. When this field is disabled, position starts from outside the page.
AnimationEffect Speed	A speed property shall define the time in seconds from the beginning to the end of the animation effect.

Table 16: OSDImage Events

Event	Short Description
EnableChanged	Occurs when Enabled property value has changed.
AnimationStart	Occurs when the animation starts.
AnimationEnd	Occurs when the animation ends.

3.3.5.1 Scaling

Sets if image will scale to component size.This is usually set through the GUI and that this value cannot be changed on runtime.

Property declaration:

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

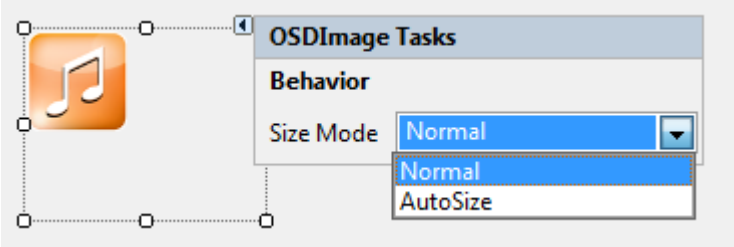
3.3.5.2 SizeMode

Controls how the osdImage will handle control sizing. By default, in Normal mode, the Image is positioned in the upper-left corner of the OSDImage, and any part of the image that is too big for the OSDImage is clipped. Using the AutoSize value causes the control to resize to always fit the image.

Property declaration:

```
public SizeMode SizeMode{get; set;}
```

This property can also be changed in canvas window as shown below:



3.3.5.3 AnimationEnabled

Gets or sets if animation will be active.This is usually set through the GUI although it may be changed at run time.

Property declaration:

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

Code window usage example:

```
//Start animating the osdImage1 component
osdImage1.AnimationEnabled = true;
```

3.3.5.4 AnimationSpeed

Gets or sets the animation speed in frames per second. This is usually set once on the GUI and not changed during run time.

Property declaration:

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

3.3.5.5 AnimationType

Gets or sets if the animation will loop continuously, bounce or run only once. See the description of the OSDAnimationTypeglobal enumeration on page 29to find out the available options here. This is usually set once on the GUI and not changed during run time.

Property declaration:

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

3.3.5.6 InitialTile

Gets or sets the initial frame from the list of images when animation is enabled. (Starts at index 0). Note that *InitialTile* must be always less than *FinalTile*, if an animation is being used. The first frame (number 0) starts counting from the left-side tile within the image inserted in the Blimp design window. This is usually set through the GUI although it may be changed at run time.

Property declaration:

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

Code window usage example:

```
osdImage1.InitialTile = 0;
```

3.3.5.7 FinalTile

Gets or sets the final tile of the animation. Note that *FinalTile*must be always less than *InitialTile*, if an animation is being used. The first frame (number 0) starts counting from the left-side tile within the image inserted in the Blimp design window. This is usually set through the GUI although it may be changed atrun time.

Property declaration:

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

Code window usage example:

```
//Even if the image has 6 frames, it is possible to define as final
frame the number 4
osdImage1.FinalTile = 4;
```

3.3.5.8 CurrentTile

Gets or sets the current tile of the component; this will show on screen whatever frame has been selected. This property can also be used to change the current displayed tile if the animation is disabled by setting Enabled to false.

Property declaration:

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

Code window usage example:

```
osdImage1.CurrentTile = 2;
```

3.3.5.9 AnimationEffectDirection

Gets or sets the directions of the Animation effect.The drop down selection list will have Left, Right, Top and Bottom.

Property declaration:

```
publicANIMATIONDIRECTION Direction{get; set;}
```

Code window usage example:

```
osdImage1.AniEffectDirection = OSD_ANIEFFECTDIRECTION.BOTTOM;
```

3.3.5.10 AnimationEffectType

Gets or sets the behavior of the Animation effect. The drop down selection list will have FadeIn, FadeOut, FlyIn, FlyOut, None. If none the animation effect will be disabled.

Property declaration:

```
publicANIMATIONEFFECTTYPE EffectType{get; set;}
```

Code window usage example:

```
osdImage1.AniEffectType = OSD_ANIEFFECTTYPE.FADEIN;
```

3.3.5.11 AnimationEffectStartPosition

A property shall specify the start position of animation effect. User will be able to enable or disable this field.If enabled, specifies the top left position of the page or component where the animation will start from (should be in x,y coordinates). When this field is disabled, position starts from outside the page.

Property declaration:

```
publicint StartPosition{get; set;}
```

Code window usage example:

```
osdImage1.AniEffectStartPositionX = 0;
osdImage1.AniEffectStartPositionY = 0;
```

3.3.5.12 AnimationEffectSpeed

A speed property shall define the time in seconds from the beginning to the end of the animation effect.

Option list as below can be selected:
4 - Very slow
3 - Slow
2 - Medium
1 - Fast
0.5- Very fast
A custom value can also be entered directly. This should be combo box /text box combination with only number. The Range should be [0.1 - 5].

Property declaration:

```
publicfloat Speed{get; set;}
```

Code window usage example:

```
osdImage1.AniEffectSpeed = 2.5;
```

3.3.5.13 EnableChanged

This event occurs when the Enabled property value 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.

Syntax: `privatevoidEnableChanged(void)`

3.3.5.14 AnimationStart

Occurs when the animation starts.

Syntax: `privatevoidAnimationStart(void)`

Code window usage example:

```
private void osdImage1_AnimationStart()
{
osdLabel1.Text = "Animation start event triggered";
}
```

3.3.5.15 AnimationEnd

Occurs when the animation ends.

Syntax: `privatevoidAnimationEnd(void)`

Code window usage example:

```
private void osdImage1_AnimationEnd()
{
osdLabel1.Text = "Animation end event triggered";
}
```

3.3.5.16 loadImageData

This method dynamically loads an image to DDR2 memory.

Method declaration:

```
publicBoolean loadImageData(UINT8[] data, UINT32 Height, UINT32 Width);
```

Paramenters:

data: raw image data buffer.

Height: Height of the raw Image data

Width: Width of the raw Image data

Code window usage example:

```
Result = osdImage1.loadImageData(buffer, 32, 32);
```

3.3.5.17 ReadImageData

This method is used to read the existing raw image data from DDR2 memory into MCU RAM. Returns true for successful read and false, if the given height/width greater than its original height/width.

Method declaration:

```
publicBoolean ReadImageData(UINT8[] data, UINT32 Height, UINT32 Width);
```

Paramenters:

data: Empty buffer to read the raw image data from DDR2 memory. Note: data buffer should have enough memory space.

Height: Height of the raw Image data

Width: Width of the raw Image data

Code window usage example:

```
Result = osdImage1.ReadImageData(buffer, 32, 32);
```

3.3.5.18 Creating an Image Animation

See Blimp user manual in section 4.4.

3.3.6 OSDProgressbar

The *OSDProgressbar*works by overlapping one foreground picture over one in the background. Depending on the value set to the bar, the 'active picture' will be elongated or cropped, giving the effect of a progress bar. This component can receive focus and has two events associated with it.

Table 17: OSDProgressbar Properties

Property or Method	Short Description
ColorDepth	Sets the color format.
Skin	Loads a skin for the progress bar.
Border	Defines overlapping image border in pixels where the active image will show in the progress bar. If All = -1, then uses Top, Bottom, Left, Right values.
EmptyImage	Gets or sets background image.
FilledImage	Gets or sets foreground image.
Orientation	Gets or sets filling direction (horizontal or vertical) of bar.
OrientationReversed	Gets or sets inversion of bar filling.
MinValue	Gets or sets the lower bound value.
MaxValue	Gets or sets the upper bound value.
Step	Gets or sets the value which the progress bar will increment and decrement.
addStep()	Adds the amount defined in property "Step" to current position.
subtractStep()	Subtracts the amount defined in property "Step" to current position.
Position	Sets or gets current position. Must be between minimum and maximum.

Table 18: OSDProgressbar Events

Event	Short Description
ValueChanged	**This event occurs whenever length of bar gets modified**
RemoteKeyPress	**This event occurs in response to a user keystroke input**

3.3.6.1 BackgroundImage

Gets or sets the background image. This is usually set through the GUI. Note that this value cannot be changed on runtime.

Property declaration:

```
publicImage BackgroundImage{get; set;}
```

3.3.6.2 ForegroundImage

Gets or sets the foreground image. This is usually set through the GUI. Note that this value cannot be changed on runtime.

Property declaration:

```
publicImage ForegroundImage{get; set;}
```

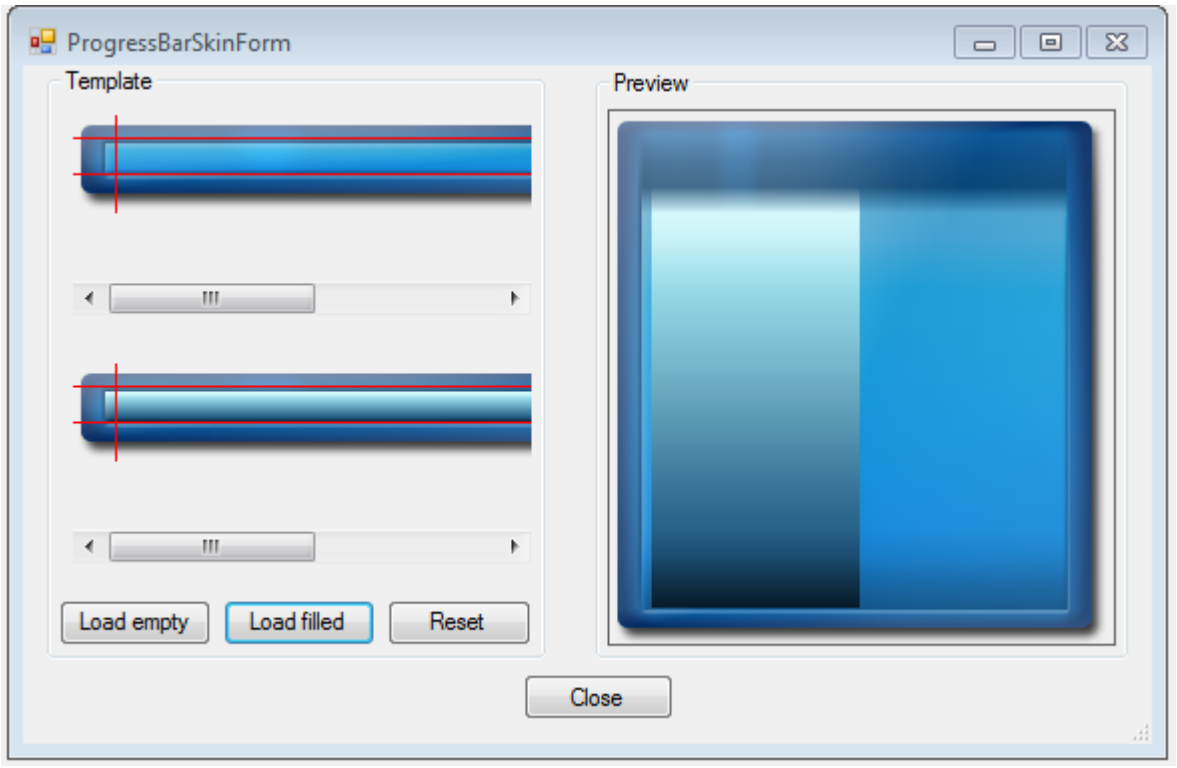
3.3.6.3 Border

Set the border width of all sides in pixels. Defines overlapping image border in pixels where the active image will show in the progress bar. If All = -1, then uses Top, Bottom, Left, Right values.This is usually set through the GUI. Note that this value cannot be changed on runtime.

Property declaration:

```
publicPadding Border{get; set;}
```

Through the skin property also we can set the Background, ForgroundImage and border. The picture shown in



Using the red color lines, we can change the border properties.

3.3.6.4 Orientation

Gets or sets the filling direction (horizontal or vertical) of the bar. This is usually set through the GUI although it may be changed on run time.

Property declaration:

```
public OSD_ORIENTATION Orientation{get; set;}
```

Code window usage example:

```
//For example, assuming that osdProgressBar1 is a horizontally-
orientated osdProgressBar component, its //orientation can be changed to
vertically-orientated during runtime through
osdProgressBar1.Orientation = OSD_ORIENTATION.VERTICAL;
```

3.3.6.5 OrientationReversed

Gets or sets the inversion of the bar filling. When vertical orientation is set, the *OSDprogressbar* is rotated 90 degree clockwise from its horizontal position. If *OrientationReversed* is set, then the *OSDProgressbar* is rotated 90 degree counterclockwise from horizontal position.

Property declaration:

```
Bool OrientationReversed{get; set;}
```

Code window usage example:

```
osdProgressBar1.OrientationReversed = true;
```

3.3.6.6 MinValue

Gets or sets the lower bound value. This is the minimum value that the bar can reach, even if the SubtractStep() method is being called.

Property declaration:

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

Code window usage example:

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

3.3.6.7 Max Value

Gets or 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;}
```

Code window usage example:

```
//Max value for contrast bar set to 100
contrastBar.MaxValue = 100;
```

3.3.6.8 Step

Gets or sets the value which the progress bar will increment and decrement. 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.

Property declaration:

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

Code window usage example:

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

3.3.6.9 addStep

Adds the amount defined in the property Step to the current position.

Method declaration:

```
public UINT32 addStep()
```

Code window usage example:

```
//Make the bar to grow up
contrastBar.addstep();
```

3.3.6.10 subtractStep

Subtracts the amount defined in the property Step to the current position.

Method declaration:

```
public UINT32 subtractStep()
```

Code window usage example:

```
//Make the bar to shorten
contrastBar.subtractstep();
```

3.3.6.11 Position

Gets or sets current position. 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;}
```

Code window usage example:

```
//Contrast bar set to 50 by default
contrastBar.Position = 50;
```

3.3.6.12 ValueChanged

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

Syntax: `privatevoidValueChanged (void)`

3.3.6.13 RemoteKeyPress

Syntax: `privatevoidRemoteKeyPress (Byte *keyCode, Boolean *cancel)`

This event occurs in response to a user keystroke input.

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 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 in Table 19. 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 tothe . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

Note that when using the hardcodedkeys, the OSDProgressbar isfilled/emptied by an amount equal to the unit defined in the *Step* property.

Table 19: Hardcoded Keys in OSDProgressbarComponent

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)

3.3.7 OSDMenubar

Through this component, an icon-based menu and submenus tree can be defined. The *OSDMenubar* uses icons to move through the different options on the menu, and allows the user to navigate through them with an animation effect. The “menu tree” is graphically defined through Blimp, as can be seen on Figure 10. This component needs to have focus in order to capture the user inputs (arrow keys). It also has six events associated to it.

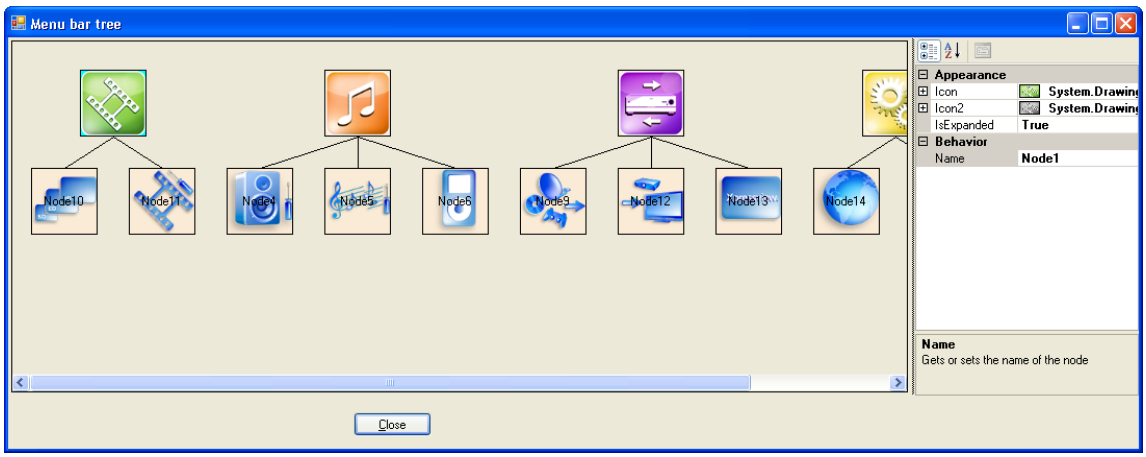


Figure 10: OSDMenubar Tree Structure Defined in Blimp

Table 20: OSDMenubar Properties.

Property	Short Description
ColorDepth	Sets Color format of the icon images
Tree	Define menubar nodes
IconPadding	Gets or sets separation (in pixels) between icons
IconSize	Gets or sets size (in horizontal * vertical pixels) of icons
IconWindow	Gets or sets position of icon which cursor is on
LeftPadding	Gets or sets separation (in pixels) between left border of component and icons
MaxIcons	Gets or sets maximum number of icons been shown on screen at same time
TopPadding	Gets or sets separation (in pixels) between top border of component and icons
Orientation	Gets or sets orientation (vertical or horizontal) of component
CanGoIn	Gets whether a sub-node is defined one level down tree, with respect to current (windowed) icon
CanGoOut	Gets whether a sub-node or root node is defined one level up tree, with respect to current (windowed) icon
CanGoNext	Gets whether a sub-node or root node is defined at right of current (windowed) icon
CanGoPrevious	Gets whether a sub-node or root node is defined at left of current (windowed) icon
CurrentState	Gets or sets global current state (root node or sub-node) in which cursor is set at
HiddenNodes	Gets or sets what nodes are set as hidden (will not be shown on *OSDMenubar*)
ScrollingSpeed	Gets or sets speed of animation when *OSDMenubar* scrolls vertical or horizontally

Table 21: OSDMenubar Events

Event	Short Description
AnimationStart	Occurs when *OSDMenubar* vertical or horizontal scrolling commences
AnimationEnd	Occurs when *OSDMenubar* vertical or horizontal scrolling finishes
StateChanged	Occurs when list has moved into new item
StateChanging	Occurs when list starts moving towards new item
StateSelected	Occurs when one of items within list is selected
RemoteKeyPress	This event occurs in response to a user keystroke input

3.3.7.1 IconPadding

Gets or sets the separation (in pixels) between the icons. This is usually set through the GUI although it may be changed on run time.

Property declaration:

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

Code window usage example:

```
osdMenubar1.IconPadding = 10;
```

3.3.7.2 IconSize

Gets or sets the size (in horizontal * vertical pixels) of the icons. Note that this value cannot be changed on runtime. All the icons within the *OSDMenuBar* have to be the same size.

Property declaration:

```
public Size IconSize{get; set;}
```

3.3.7.3 IconWindow

Gets or sets the position of the icon which the cursor is on. The icon which has this "focus" is the only one which the user could navigate into to move through its submenus.

Property declaration:

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

Code window usage example:

```
osdMenubar1.IconWindow = 2;
```

3.3.7.4 LeftPadding

Gets or sets the separation (in pixels) between the left border of the component and the icons. This is usually set through the GUI although it may be changed on run time.

Property declaration:

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

Code window usage example:

```
osdMenubar1.LeftPadding = 20;
```

3.3.7.5 MaxIcons

Gets or sets the maximum number of icons being shown on the screen at the same time. This is usually set through the GUI although it may be changed on run time.

Property declaration:

```
public int UINT32 MaxIcons{get; set;}
```

Code window usage example:

```
osdMenubar1.MaxIcons = 5;
```

3.3.7.6 TopPadding

Gets or sets the separation (in pixels) between the top border of the component and the icons. This is usually set through the GUI although it may be changed on run time.

Property declaration:

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

Code window usage example:

```
osdMenubar1.TopPadding = 20;
```

3.3.7.7 Orientation

Gets or sets the orientation (vertical or horizontal) of the component. This is usually set through the GUI and that this value cannot be changed on runtime. Note that, when in the emulator window, the keys that allow the user to move through the menubar will change with the orientation. In a vertical orientation, the up and down arrows are used to move through the items in the same tree level. Right and left arrows are used to, respectively, descend and ascend into the tree hierarchy. When in horizontal position, right and left keys are used to move through the items in the same tree level. Down and up arrows are used to, respectively, descend and ascend into the tree hierarchy. These keys are hardcoded and cannot be user selected.

Property declaration:

```
public OSD_ORIENTATION Orientation{get; set;}
```

3.3.7.8 CanGoIn

Gets whether or not a sub-node is defined one level down the tree, with respect to the current (windowed) icon.

Property declaration:

```
public bool CanGoIn{get}
```

Code window usage example:

```
//Show a small arrow close to the icon indicating the direction the user
can scroll to
arrowRight.Visible = (osdMainMenu.CanGoIn);
```

3.3.7.9 CanGoOut

Gets whether a sub-node or root node is defined one level up the tree, with respect to the current (windowed) icon.

Property declaration:

```
public bool CanGoOut{get}
```

Code window usage example:

```
//Show a small arrow close to the icon indicating the direction the user
can scroll to
arrowLeft.Visible = (osdMainMenu.CanGoOut);
```

3.3.7.10 HiddenNodes

Gets or sets what nodes are set as hidden (these will not be shown on the OSDMenubar). This property is useful to hide some options not available for certain regions when some functionality is not enabled. Note that this property can only be set in the code window, and not in the Project Navigator panel.

Property declaration:

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

Code window usage example:

```
//Do not show nodes 2 and 5.
osdMenuBar.HiddenNodes[2] = true;
osdMenuBar.HiddenNodes[5] = true;
```

3.3.7.11 ScrollingSpeed

Gets or sets the speed of the animation when OSDMenubar scrolls vertical or horizontally. Speed is defined in pixels/frame. A value of '0' disables the animation.

Note that ScrollingSpeed having limits and it depends upon the IconPadding and IconSize property values. If *OSDMenubar* orientation is horizondal then the ScrollingSpeed should be <= IconWidth + IconPadding. If *OSDMenubar* orientation is vertical then the ScrollingSpeed should be <= IconHeight + IconPadding.

Property declaration:

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

Code window usage example:

```
osdMenuBar.ScrollingSpeed = 5;
```

3.3.7.12 CanGoNext

Gets whether a sub-node or root node is defined at the right of the current (windowed) icon.

Property declaration:

```
public bool CanGoNext{get}
```

Code window usage example:

```
//Show a small arrow close to the icon indicating the direction the user
can scroll to
arrowDown.Visible = (osdMainMenu.CanGoNext);
```

3.3.7.13 CanGoPrevious

Gets whether a sub-node or root node is defined at the left of the current (windowed) icon

Property declaration:

```
public bool CanGoPrevious{get}
```

Code window usage example:

```
//Show a small arrow close to the icon indicating the direction the user
can scroll to
arrowUp.Visible = (osdMainMenu.CanGoPrevious);
```

3.3.7.14 CurrentState

Gets or sets the global current state (root node or sub-node) at which the cursor is set. The numbering order for the root nodes and sub-nodes is set by the initial creation of the tree. The numbering is set from left to right, following the node and then the sub-nodes (if any). Figure 11 shows this concept.

Property declaration:

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

Code window usage example:

```
//User can pool the currentstate to display the different options of the
menu and submenus
if (osdMainMenu.CurrentState == 10) {…
```

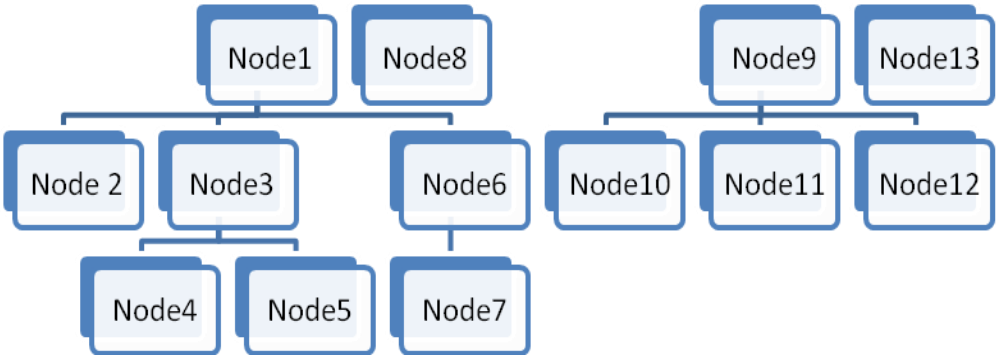


Figure 11: Numbering Scheme for Current State

3.3.7.15 AnimationStart

This event occurs when the *OSDMenubar* vertical or horizontal scrolling commences.

Syntax: `privatevoidAnimationStart(void)`

3.3.7.16 AnimationEnd

This event occurs when the *OSDMenubar* vertical or horizontal scrolling finishes.

Syntax: `privatevoidAnimationEnd(void)`

3.3.7.17 StateChanged

This event occurs when the list has moved into a new item. The event is triggered just in the moment in which the moving menu bar gets to the icon and stops its movement. Note how this event always triggers after *StateChanging* event.

Syntax: `privatevoidStateChanged(void)`

3.3.7.18 StateChanging

This event occurs when the list starts moving towards the new item that is, just in the moment in which the user gives the order to move to a new item on the list.

Syntax: `privatevoid StateChanging( OSD_MENUBAR_FLOW flow, Byte* targetChild)`

Parameters:

`flow`:Gives information regarding the direction in which the user is moving to through the *OSDMenuBar*respect the current item (*currentState*). It is an enum declared as:

```
public enum OSD_MENUBAR_FLOW
    {
        MB_PREV,
        MB_NEXT,
        MB_OUT,
        MB_IN
    }
```

`MB_PREV`: User is moving towards the next item in the same hierarchical node level.

`MB_NEXT`: Useris moving towards the previous item in the same hierarchical node level.

`MB_OUT`: Useris moving towards a higher hierarchical level, that is, ascending one level on the menu tree.

`MB_IN`: Useris moving towards a lower hierarchical level, that is, descending one level on the menu tree.

For example, if the cursor (that is, currentState) is at Node3 in Figure 11, and the user presses (on a vertical-orientated *OSDMenuBar*) the right key, "flow" enumeration variable will be equal to OSD_MENUBAR_FLOW.MB_IN;

`*targetChild`: Allows the user to define what is going to be the destination item to which the menu bar is going to move. By default, when going into the hierarchy tree, the next selected item will be the one following the numbering order of the tree. For example, if the cursor (that is, currentState) is at Node3 in Figure 11 and presses the right key, the menu moves to Node4. By using "targetchild" this behavior can be changed. This can be useful when the user wants the menu to remember the last item selected within a submenu.

Code window usage example:

```
{

int previousState;

    public void Load()
    {
        OsdApi.ADIAPI_OSDEgSetFocusComponent(osdMainMenu);
```

```
    }
    public void Dispose()
    {
    }

    private void osdMainMenu_StateChanging(OSD_MENUBAR_FLOW flow,Byte*
targetChild)
    {
if ((flow == OSD_MENUBAR_FLOW.MB_IN) && (previousState == 3))
        {
            *targetChild = 3;
        }

        previousState = osdMainMenu.CurrentState;
    }
}
```

3.3.7.19 StateSelected

This event occurs when the user navigates to the last item within the submenu tree and selects it (that is, presses the *right* key on a vertically-orientated menubar, or *down* on a horizontally-orientated one). Usually, the user would use this event to give focus to an *OSDListbox* component which would allow the user to make changes to the feature selected through the menu bar.

Syntax: `privatevoidStateSelected(void)`

Example:

```
 private void osdMainMenu_StateSelected()
    {

        // Switch focus to the list box
        OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListBox1);
        osdListBox1.DefaultTextColor = 0xFFE0E0E0u;
        osdListBox1.SelectedTextColor = 0xFFE0E0E0u;
    }
```

3.3.7.20 RemoteKeyPress

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

Syntax: `private void RemoteKeyPress (Byte *keyCode, Boolean *cancel)`

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 component will not receive the keystroke (for example, an `OSDMenubar` will not receive the "down" key).

There are keys that automatically interact with the OSDMenubar 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 22.

When 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 . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

Table 22: Hardcoded Keys in OSDMenubar Component

Key	Key Code (decimal)	Function
Up Arrow	38	Scrolls icons downwards (effectively selecting the icon above the actual position)
Down Arrow	40	Scrolls icons upwards (effectively selecting the icon below the actual position)
Left Arrow	37	Ascends in hierarchy of *OSDMenuBar*
Right Arrow	39	Descends in hierarchy of *OSDMenuBar*

Code window usage example:

```
//Force the OSdMenubar to scroll in only one direction:
private void osd_menubar_Page_RemoteKeyPress(Byte* keyCode, Boolean*
cancel)
{
    if (*keyCode == 0x28)
    {
        *cancel = true; //Cancel down arrow keystroke
    }
}


//Invert the up/down axis:
private void osd_menubar_Page_RemoteKeyPress(Byte* keyCode, Boolean*
cancel)
{
    if (*keyCode == 0x28)
    {
        *keyCode = 0x26; //Swap Down keystroke by Up keystroke
    }

    else if (*keyCode == 0x26)
    {
        *keyCode = 0x28; //Swap Up keystroke by Down keystroke
    }
}
```

3.3.7.21 Creating a Menu Bar

Follow these steps to create a menu bar.

- Drag and drop an OSDMenubar component into the Designer Canvas.
- Use the Property Navigator to access the submenu used to define the icon tree. (The property is called ‘Tree’ as shown in Figure 12.)

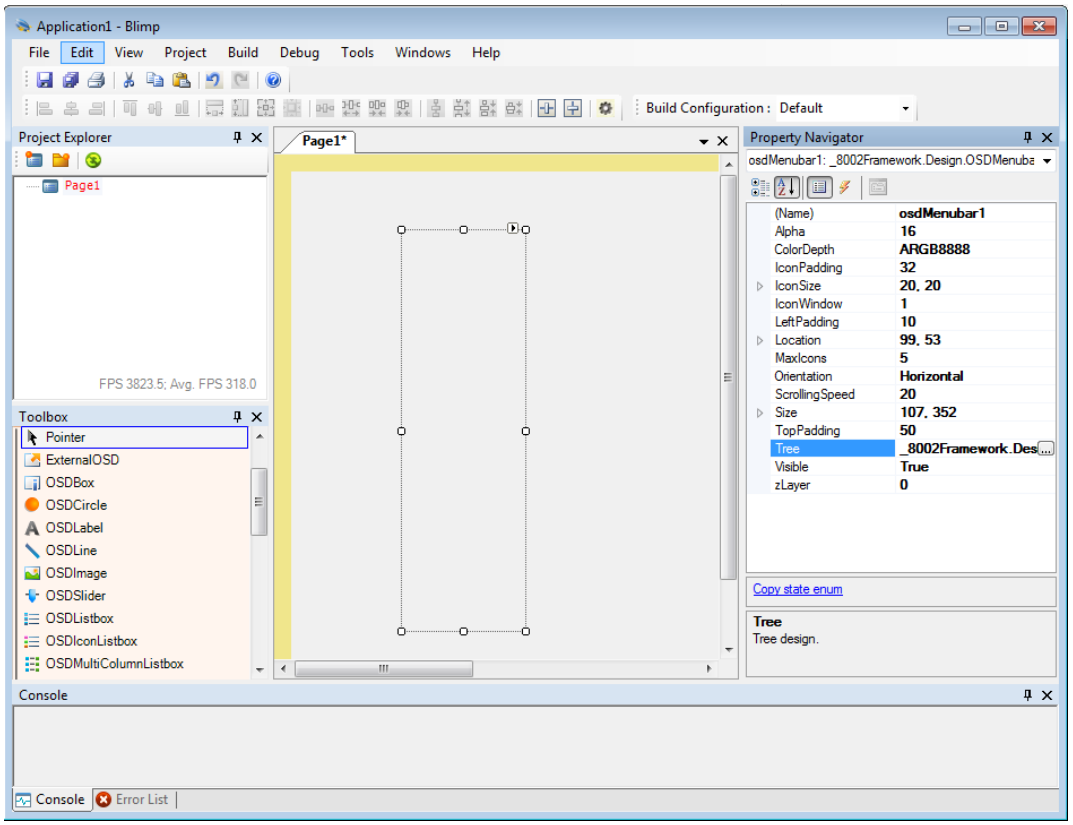


Figure 12: Inserting OSDMenubar Component into Design Window

- Right click on the Menubar Tree left panel and select add root-node to create the main menu item, Node1. This will appear as an empty box named Node1, as shown in Figure 13. The name can be changed from the Name property. Another root-node can be created by right clicking on the panel.

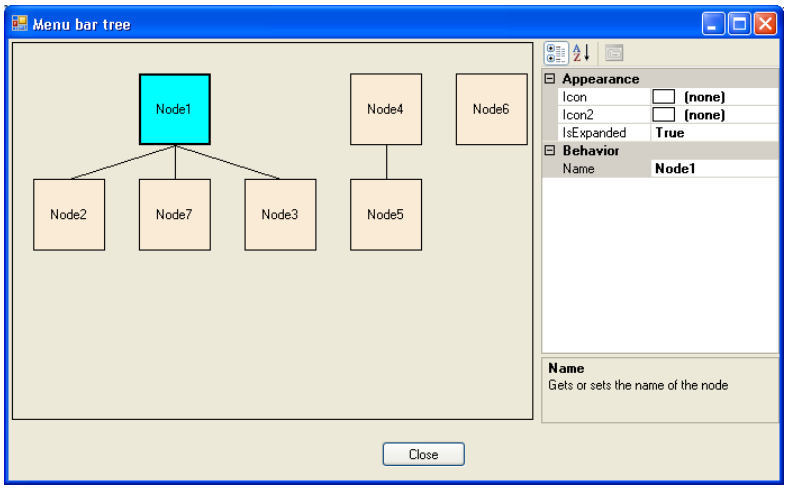


Figure 13: Defining OSDMenubar Tree

- Right click on Node1 and click on Add node to create a submenu node. Node2 will appear under Node1. There is no limit to the number of root nodes or subnodes.

  An example of a menu structure is shown in Figure 13.

Adding images to each menu items:

Follow this step to assign icons to nodes in the Tree window.

- In property, select Icon and click on the box with three dots to select an active image.
- In property, select Icon2 and click on the box with three dots to select an inactive image.

Each node has 2 icons; one to be shown when the icon is active, and the other to be shown when it is not active. For example, a grayed version of the icon could be selected to be shown when the icon is not active.

If a non active icon is not used, the active icon will be shown at all times. Both icons are selected, respectively, through “Icon” and “Icon2” properties of the panel on the right. Once the icons have been assigned to each node, the window can be closed.

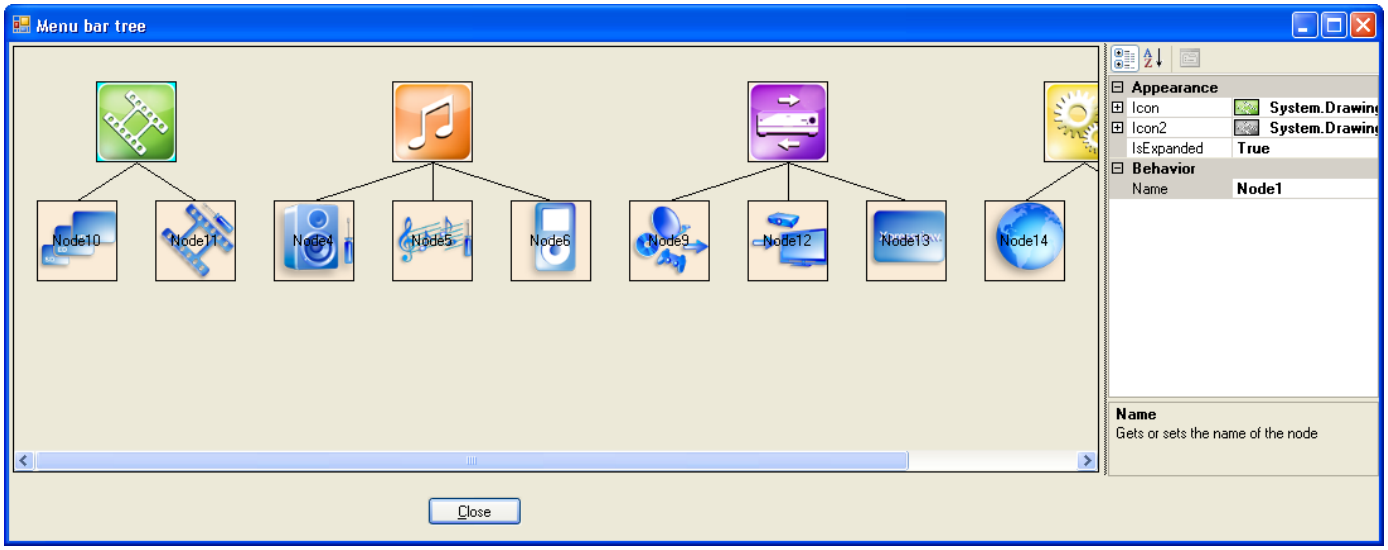


Figure 14: Finished OSDMenubar Tree

The root node icons of the *OSDMenubar* component are now visible in the *Designer Canvas*. Note that, since this component does not define any background image, you may want to place an *OSDImage* component behind the inserted menubar.

You can now define a scrolling speed. This sets up the animation speed (defined in pixels/frame) that the component exhibits when doing both vertical and horizontal scrolling, that is, when navigating through the icons. A value of '0' disables any animation.

The *OSDMenubar*is readynow to be navigated through, just by setting the focus property to it (see the following code) and running the emulator.

```
OsdApi.ADIAPI_OSDEgSetFocusComponent(osdMenuBar);
```

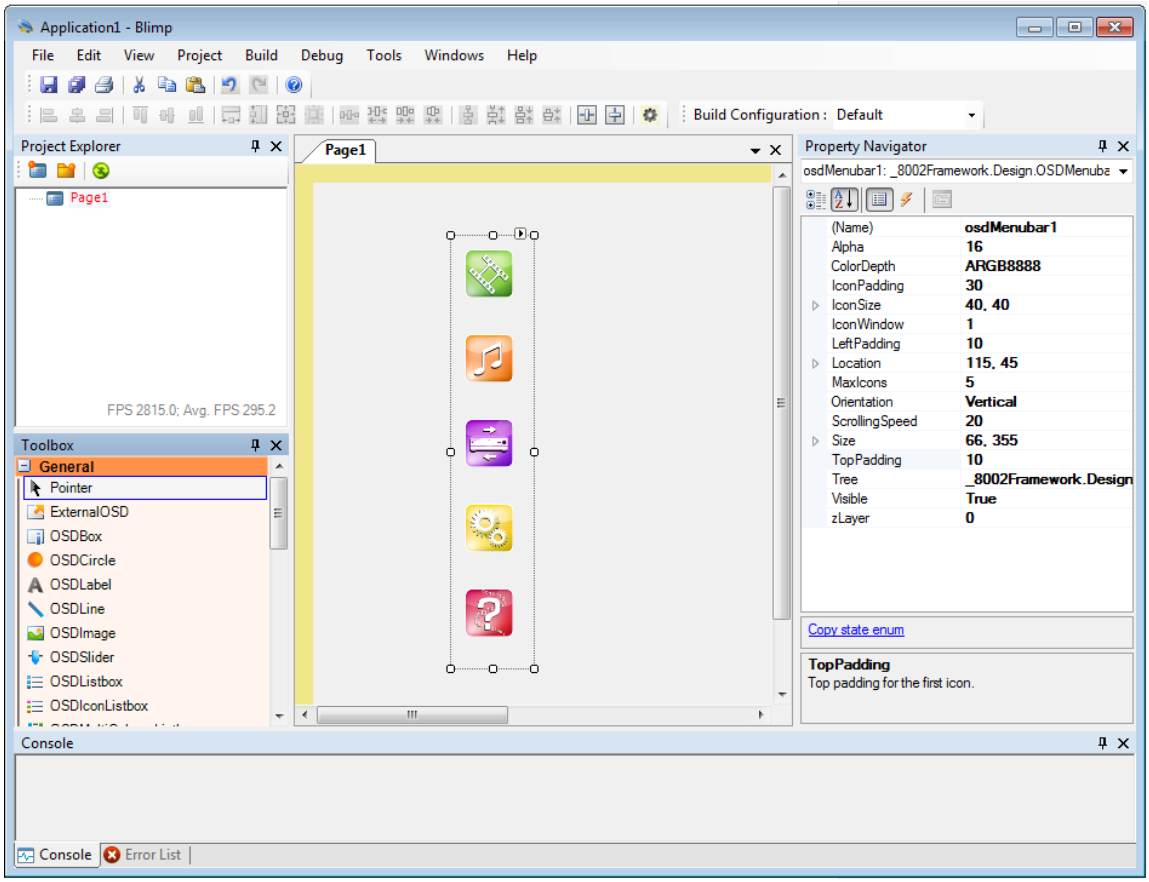


Figure 15: Complete OSDMenubar Tree Inserted in Design Window

- Blimp provides a function that facilitates the integration of the OSDMenuBar component into the rest of the scripting code.

  Normally, you create an enum type in the code to define each node of the menu. Since this enumeration can become quite big and tedious to code, you can select Copy state enum in the Property Navigator panel to copy the declaration of this enum into the clipboard.

  The enumeration name will be "<name given to the OSDMenubar>_states". The members of the *enum* are ordered as per the scheme illustrated Figure 11.

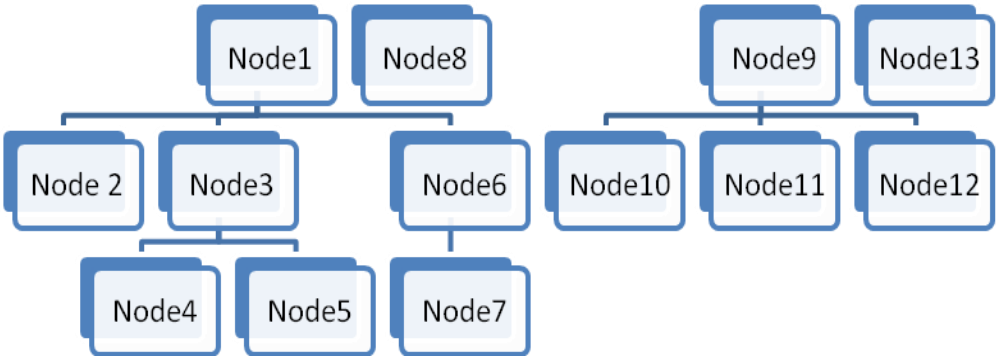


Figure 16: Enumeration numbering Scheme for menubar nodes

For example, if there is an *OSDMenubar* called *osdMenubar*, as shown in Figure 13, the following code is copied into the clipboard.

```
public enum osdMenubar_states {
        Node1 = 1,
        Node2 = 2,
        Node7 = 3,
        Node3 = 4,
        Node4 = 5,
        Node5 = 6,
        Node6 = 7
    }
```

3.3.8 OSDTextbox

This component is a box which allows the user to input alphanumeric data. This text input can be done in several ways. The normal mode is to use the arrow keys, to scroll up or down through a list of different characters. Another mode of text inputting is to use a writing system similar to the one used in mobile phones. The last mode in which data can be input is through an *OSDKeyboard* component. For the first two modes, customizable characters tables can be defined, so user input can be restricted to those characters.

The *OSDTextbox* component may have focus and has three events associated.

Table 23: OSDTextbox Properties

Property or Method	Short Description
Text	Sets text within component.
FgColor	Gets or sets foreground color of text.
CursorColor	Gets or sets color of cursor.

HighlightedCharColor	Gets or sets color of highlighted character.
TextAlign	Gets or sets alignment of text within *OSDTextbox* component.
Font	Gets or sets font being used on *OSDTextbox* component.
MaxLength	Gets or sets maximum number of Unicode characters the *OSDTextbox* can contain at any time.
BlinkingSpeed	Sets or gets cursor blinking speed.
ForceCursor	Gets or sets if cursor will always be visible.
KeyTable	Gets or sets current key table.
MaxChars	Sets maximum number of characters which can be added to *OSDTextbox* component.
MinChars	Sets minimum number of characters which *OSDTextbox* component must contain.
PasswordCharacter	Gets or sets character that will be used for masking text.
PasswordCharacterCode	Defines character code that will be used for masking text.
PasswordPreview	Gets or sets, when password mode is enabled, if last entered character will be password-masked.
RollbackEnabled	Sets if cursor will roll back from end to beginning and from beginning to end when pressing left and right arrows.
UsingPhoneMode	Gets or sets if text input behavior is like a phone.
appendChar()	Adds a character to current position of cursor.
delChar()	Removes a character from current position of cursor.
moveCursorPrev()	Moves cursor to previous character (to left).
moveCursorNext()	Moves cursor to next character (to right).
setTextFormat()	Sets formatted text within textbox.
readText()	Reads the text from textbox in ascii format and store into buffer.
readTextW()	Reads the text from textbox in unicode format and store into buffer.

Table 24: OSDTextbox Events

Event	Short Description
RemoteKeyPress	Occurs in response to user keystroke input.
ValueChanged	Occurs when text within *OSDTextbox* is changed.
CursorPositionChanged	Occurs when the cursor position has changed.

3.3.8.1 Text

Sets the text within the component. Although usually the textbox is used to receive the user text input, setting a text directly into the *OSDTextbox* component is also possible (as it is done on the *OSDLabel* component). This would be useful, for example, when editing a listbox item if you want to show the text the user can now modify.

Property declaration:

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

Code window usage example:

```
osdTextbox.Text = "Please write your name here";
```

3.3.8.2 FgColor

Gets or sets the foreground color of the text.

Property declaration:

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

Code window usage example:

```
//White color
osdTextbox.FgColor = 0xFFFFFFFFu;
```

3.3.8.3 CursorColor

Gets or sets the color of the cursor.

Property declaration:

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

Code window usage example:

```
//White Color
Ipaddress1.CursorColor = 0xFFFFFFFFu;
```

3.3.8.4 HighlightedCharColor

Gets or sets the color of the highlighted character.

Property declaration:

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

Code window usage example:

```
//Gold Color
Ipaddress1.HighlightedCharColor = 0xFFFFD700u;
```

3.3.8.5 TextAlign

You can use this property to align the text within an *OSDTextbox* to match the layout of controls on your page. 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, BOTTOMRIGHT) and the text will be aligned with the right edge of the labels to align with your controls. This is usually set through the GUI although it may be changed on run time.

Property declaration:

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

Code window usage example:

```
osdTextbox.TextAlign = OSD_CONTENTALIGNMENT.TOPRIGHT;
```

3.3.8.6 Font

Gets or sets the font being used on the *OSDTextbox* component.

For more details please refer the section Selecting fonts.

Code window usage example:

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

3.3.8.7 setTextFormat

Sets formatted text within the text box. This method is very similar to the *sprintf* function in C. It is used to deal with the string concatenation and representation that needs to be done in an ANSI-C compatible fashion. Refer Table 10 list of supported formats.

Method declaration:

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

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

Code window usage example:

```
osdTextbox.setTextFormat("HDMI Input %d Selected",inputnumber);
```

3.3.8.8 readTextW

Gets the text of any *OSDTextbox* and stores it into a buffer. The read text can be any Unicode character.

Method declaration:

```
publicvoidreadTextW(ushort* buffer, ushort size);
```

buffer: Array where the read characters are going to be stored in. It should be long enough to hold the value set in *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) which are going to be read from the item. For example, if the item string length is 11, size should be set to 12.

Code window usage example:

```
ushort[] buffer = new ushort[15];

osdTextbox1.Text = "First Item";
//Read 11 characters from osdTextbox1 and put the read string into buffer
osdTextbox1.readTextW(buffer,11);
osdLabel1.TextW = buffer;    // sets the text buffer to osdLabel1
```

3.3.8.9 readText

Gets the text of any *OSDTextbox* ascii format and stores it into a buffer or reads the text of *OSDTextbox* from ddr2 and writes inside the buffer

Method declaration:

```
publicvoidreadText(CHAR* buffer, ushort size);
```

buffer: Array where the read characters are going to be stored in. It should be long enough to hold the value set in *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) which are going to be read from the item. For example, if the item string length is 11, size should be set to 12.

Code window usage example:

```
char[] buffer = new char[15];
osdTextbox1.Text = "First Item";
```

```
//Read 11 characters from osdTextbox1 and put the read string into
buffer
osdTextbox1.readText(buffer,11);
osdLabel1.TextW = buffer;    // sets the text buffer to osdLabel1
```

3.3.8.10 MaxLength

Gets or sets the maximum number of Unicode characters the *OSDTextbox* can contain at any time.
Note that this property availableonly property navigator of Blimp and not to the scripting page.

Method declaration:

```
publicUINT16 MaxLength{get; set;}
```

3.3.8.11 BlinkingSpeed

Sets or gets the cursor blinking speed. The Blinkingspeed is defined in frames.

Method declaration:

```
publicUINT8 BlinkingSpeed(get; set;)
```

Code window usage example:

```
osdTextbox.BlinkingSpeed = 3;
```

3.3.8.12 ForceCursor

Gets or sets if the cursor will always be visible, even if focus is not set to the component. This property is useful when the *OSDTextbox* is working with an *OSDKeyboard*(so the textbox has no focus) so the cursor can be visible.

Method declaration:

```
publicBool ForceCursor(get; set;)
```

Code window usage example:

```
//The cursor will be visible even if the component does not have focus

osdTextbox.ForceCursor = true;
```

3.3.8.13 KeyTable

Gets or sets the current key table.

Property declaration:

```
publicUINT32 KeyTable {get; set;}
```

Code window usage example:

```
//Use the character set defined in row 1
osdTextbox1.KeyTable = 1;
```

3.3.8.14 MaxChars

Sets the maximum number of characters which can be added to the *OSDTextbox* component. This means that it is only possible to add characters (either using normal/phone mode or add*Char*() method) if there are less than *MaxChars*characters within the *OSDTextbox*.

Property declaration:

```
publicUINT32 MaxChars {get; set;}
```

Code window usage example:

```
//Set a maximum of 10 characters to the textbox
osdTextbox1.MaxChars = 10;
```

3.3.8.15 MinChars

Sets the minimum number of characters which the *OSDTextbox* component must contain. This means that it is only possible to delete characters (either using the backspace key or *delChar*() method) if there are more than *MinChars*characters remaining within the *OSDTextbox*.

Property declaration:

```
UINT32 MinChars {get; set;}
```

Code window usage example:

```
//Set a maximum of 5 characters to the textbox
osdTextbox1.MinChars = 5;
```

3.3.8.16 PasswordCharacter

Gets or sets the character to be used for masking the text. Any Unicode character code may be used. Note that *PasswordCharacter* and *PasswordCharacterCode* are complementary properties; setting one also sets the other.

Property declaration:

```
UINT32 PasswordCharacter {get; set;}
```

Code window usage example:

```
//Use '*' as the password character
osdTextbox1.PasswordCharacter= 42;
```

3.3.8.17 PasswordCharacterCode

Defines the character code to be used for masking the text. Any Unicode character code may be used. Note that *PasswordCharacter* and *PasswordCharacterCode* are complementary properties; setting one also sets the other.

Note that this property is not available from within the script window.
it only can be set from the *Property Navigator* in the GUI.

3.3.8.18 PasswordPreview

Gets or sets, when password mode is enabled, if the last entered character is password-masked.

Property declaration:

```
publicUINT32 PasswordPreview {get; set;}
```

Code window usage example:

```
// Show last entered character when in password-protected mode
osdTextbox1.PasswordPreview= true;
```

3.3.8.19 RollbackEnabled

Gets or sets if rollback is enabled. If enabled, when moving the cursor to the end of the string, it appears at the other end.

This property usually set once in blimp property GUI and this value
cannot be changed on runtime.

Property declaration:

```
publicBool Rollback {get; set;}
```

3.3.8.20 UsingPhoneMode

Gets or sets if the text input behavior is like a phone

Property declaration:

```
publicBool UsingPhoneMode {get; set;}
```

Code window usage example:

```
osdTextbox1.UsingPhoneMode = true;
```

3.3.8.21 appendChar

Adds a character to the current position of the cursor.

Method declaration:

```
publicvoidappendChar(ushort character)
```

Code window usage example:

```
// below example receives the key from page1 remotekeypress event and
append the char to osdTextbox1

 private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{

osdTextbox1.appendChar((ushort)*keyCode);

}
```

3.3.8.22 delChar

Removes a character from the current position of the cursor.

Method declaration:

```
publicvoiddelChar(void)
```

Code window usage example:

```
// below example delete character from current position of cursor when
backspace key is pressed

private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{

  if(*keyCode == 8) {// backspace keycode
   osdTextbox1.delChar();
   }

}
```

3.3.8.23 moveCursorPrev

Moves the cursor to the previous character (to the left).

Method declaration:

```
public void moveCursorPrev(void)
```

Code window usage example:

```
// below example moves the cursor from current position to previous when
left key is pressed

private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{
     if(*keyCode == 37)osdTextbox1.moveCursorPrev();
}
```

3.3.8.24 moveCursorNext

Moves the cursor to the next character (to the right).

Method declaration:

```
public void moveCursorNext(void)
```

Code window usage example:

```
// below example moves the cursor from current position to next when
right key is pressed

private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{
osdTextbox1.moveCursorNext();
}
```

3.3.8.25 RemoteKeyPress

Syntax: `private void 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 `OSDTextBox`does not receive the “right” key.

There are keys that automatically interact with the*OSDTextBox*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 25.

When 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 . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

The*OSDTextBox*is a special component in the sense that it may have different hardcoded keys depending on the mode of operation, as shown in Table 25 and Table 26.

Table 25: Hardcoded Keys in OSDTextBoxComponent for Normal Mode of Operation

Key	Key Code (Decimal)	Function
Left Arrow	37	Move cursor to left
Right Arrow	39	Move cursor to right
Up Arrow	38	Scroll up through character list as defined in entry table
Down Arrow	40	Scroll down through character list as defined in entry table
Backspace	8	Delete character

Table 26: Hardcoded Keys in OSDTextBoxComponentfor Phone Mode of Operation

Key	Key Code (Decimal)	Function
Left Arrow	37	Move cursor to left
Right Arrow	39	Move cursor to right
Backspace	8	Delete character
Number key 0	48	Press repeatedly to move through characters defined in entry table
Number key 1	49	Press repeatedly to move through characters defined in entry table

Number key 2	50	Press repeatedly to move through characters defined in entry table
Number key 3	51	Press repeatedly to move through characters defined in entry table
Number key 4	52	Press repeatedly to move through characters defined in entry table
Number key 5	53	Press repeatedly to move through characters defined in entry table
Number key 6	54	Press repeatedly to move through characters defined in entry table
Number key 7	55	Press repeatedly to move through characters defined in entry table
Number key 8	56	Press repeatedly to move through characters defined in entry table
Number key 9	57	Press repeatedly to move through characters defined in entry table

3.3.8.26 ValueChanged

This event occurs when the text within the *OSDTextbox* is changed.

Syntax: `private void ValueChanged(void)`

3.3.8.27 CursorPositionChanged

This event occurs when the cursor position changed within text in the *OSDTextbox*.

Syntax: `private void CursorPositionChanged(void)`

3.3.8.28 Creating a OSDTextbox

See Blimp user manual in section 4.5.

3.3.9 OSDIptextbox

This component is basically a box which allows the user to input an IP address through the remote controller (by using up/down keys to scroll through the 0-9 numbers) or *OSDKeyboard* (or assign it directly through the code). This component has focus and three associated events.

Table 27: OSDIpTextbox Properties

Property	Short Description
IPAddr	Gets or sets IP address within component.
FgColor	Gets or sets foreground color of text.
CursorColor	Gets or sets color of cursor.
HighlightedCharColor	Gets or sets color of highlighted character.
Font	Gets or sets font being used on *OSDIptextbox* component.
BlinkingSpeed	Sets or gets cursor blinking speed.

ForceCursor	Gets or sets if cursor will always be visible.

Table 28: OSDIpTextbox Events

Event	Short Description
RemoteKeyPress	Occurs in response to an user keystroke input.
ValueChanged	Occurs when text within OSDIptextbox is changed.
CursorPositionChanged	Occurs when the cursor position has changed.

3.3.9.1 IPAddr

Gets or sets the IP address within the component.

Property declaration:

```
publicUINT32 IpAddr {get; set;}
```

Code window usage example:

```
//Set IP Address to 128.168.0.1
Ipaddress1.IPAddr = 0x80A80001;
```

3.3.9.2 FgColor

Gets or sets the foreground color of the text.

Property declaration:

```
publicUINT32 FgColor {get; set;}
```

Code window usage example:

```
//White Color
Ipaddress1.FgColor = 0xFFFFFFFFu;
```

3.3.9.3 CursorColor

Gets or sets the color of the cursor.

Property declaration:

```
publicUINT32 CursorColor {get; set;}
```

Code window usage example:

```
//Black Color
Ipaddress1.CursorColor = 0xFF00000u;
```

3.3.9.4 HighlightedCharColor

Gets or sets the color of the highlighted character.

Property declaration:

```
publicUINT32 HighlightedCharColor {get; set;}
```

Code window usage example:

```
   //Gold Color
Ipaddress1.HighlightedCharColor = 0xFFFFD700u;
```

3.3.9.5 TextAlign

You can use this property to align the text within an *OSDIptextbox* to match the layout of controls on your page. 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, BOTTOMRIGHT) and the text will be aligned with the right edge of the labels to align with your controls. This is usually set through the GUI although it may be changed on run time.

Property declaration:

```
publicOSD_CONTENTALIGNMENT TextAlign {get; set;}
```

Code window usage example:

```
Ipaddress1.TextAlign = OSD_CONTENTALIGNMENT.TOPRIGHT;
```

3.3.9.6 Font

Gets or sets the font being used on the *OSDIptextbox* component.

For more details please refer the section Selecting fonts.

Code window usage example:

```
Ipaddress1.Font = new Font("Arial", 15f, FontStyle.Regular);
```

3.3.9.7 BlinkingSpeed

Sets or gets the cursor blinking speed. The Blinkingspeed is defined in frames.

Method declaration:

```
publicUINT8 BlinkingSpeed(get; set;)
```

Code window usage example:

```
Ipaddress1.BlinkingSpeed = 3;
```

3.3.9.8 ForceCursor

Gets or sets if the cursor will always be visible, even if focus is not set to the component. This property is useful when the *OSDIptextbox* is working with an *OSDKeyboard*(so the textbox has no focus) so the cursor can be visible.

Method declaration:

```
publicBool ForceCursor(get; set;)
```

Code window usage example:

```
//The cursor will be visible even if the component does not have focus

Ipaddress1.ForceCursor = true;
```

3.3.9.9 CursorPosition

Gets or sets the cursor position of *OSDIptextbox.*

Method declaration:

```
Bool CursorPosition(get; set;)
```

Code window usage example:

```
//The cursor will be visible even if the component does not have focus

Ipaddress1.CursorPosition = 2;
```

3.3.9.10 RemoteKeyPress

Syntax: `private void RemoteKeyPress(Byte keyCode, Boolean *cancel)`

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

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 component will not receive the keystroke (e.g. an `OSDIptextBox`will not receive the “right” key).

There are keys that automatically interact with the *OSDIptextBox* 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 29.

When 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 . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

Table 29: Hardcoded Keys in OSDIptextBox Component

Key	Key Code (Decimal)	Function
Left Arrow	37	Move cursor to left
Right Arrow	39	Move cursor to right
Up Arrow	38	Scroll up through numbers 0 to 9
Down Arrow	40	Scroll down through numbers 9 to 0

3.3.9.11 ValueChanged

This event occurs when the text within the *OSDIptextbox* is changed.

Syntax: `private void ValueChanged(void)`

3.3.9.12 CursorPositionChanged

Occurs when the cursor position has changed.

Syntax: `private void CursorPositionChanged(void)`

3.3.10 OSDKeyboard

The *OSDKeyboard* component displays a virtual keyboard on the screen which can be used to write Unicode text into an *OSDTextbox* component, although it can also be used to send text to any other text-based OSD component, string or variable. This component may receive focus and defines one property and four events.

Prior to placing this component on the design window, the user needs to create a keyboard template. Through this template, the user defines the look and behavior of the keyboard which later will be placed within the *OSDKeyboard* component. This process is explained in Section3.3.10.6.

The *OSDKeyboard* is normally used to input text to an *OSDTextbox*component.

Table 30: OSDKeyboard Properties

Property	Short Description
Template	Assigns a keyboard template to *OSDKeyboard* component.
ColorDepth	Color format of the icon images.

Table 31: OSDKeyboard Events

Event	Short Description
HighlightedKeyChanging	Occurs when user starts to move from one virtual keyboard key to other.
HighlightedKeyChanged	Occurs when move from one virtual keyboard key to other has finished.
KeySelected	Occurs when a key within virtual keyboard is pressed.
RemoteKeyPress	Occurs in response to user keystroke input.

3.3.10.1 Template

Assigns a keyboard template to the *OSDKeyboard* component. Usually, the template is assigned through the *Property Navigator*, and it is not needed to be changed during runtime.

Property declaration:

```
KeyboardTemplate Template {get; set;}
```

Code window usage example:

```
//Change template on runtime
osdKeyboard1.Template = KeyboardTemplateManager.keyboard_template;
```

3.3.10.2 HighlightedKeyChanging

Syntax: `private void HighlightedKeyChanging (UInt16 currentKey, UInt16* targetKey)`

This event occurs when the user starts to move from one virtual keyboard key to another.

Parameters:

`currentKey`: Value of the current key at the moment the user gave the order of moving to a different virtual keyboard key.

`*targetKey`: Value of the destination key towards which the user decided to move.

3.3.10.3 HighlightedKeyChanged

Syntax: `private void HighlightedKeyChanged (UInt16 keycode)`

This event occurs when the move from one virtual keyboard key to another has finished.

Parameters:

keycode: Value of the key

3.3.10.4 KeySelected

Syntax: `private void KeySelected (Uint16 keycode)`

This event occurs when a key within the virtual keyboard is pressed.

Parameters:

keycode: Value of the pressed key.

Code window usage example:

```
//The event is used to capture the keys from the virtual keyboard and
send them to an OSDTextBox.

private void osdKeyboard1_KeySelected(UInt16 keyCode)
{
    if (keyCode == 64)              //Delete key within the keyboard
     {
         osdTextbox1.delChar();
    }
    else if (keyCode == 45)         //Left arrow key within the keyboard
    {
        osdTextbox1.moveCursorPrev();
    }
    else if (keyCode == 94)         //Right arrow key within the keyboard
    {
        osdTextbox1.moveCursorNext();
    }
    else
    {
//Any other key, append it to the textbox
       osdTextbox1.appendChar(keyCode);
}
}
```

3.3.10.5 RemoteKeyPress

Syntax: `private void RemoteKeyPress( Byte *keycode, Boolean *cancel)`

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

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 component will not receive the keystroke, for example, an `OSDKeyboard` will not receive the "right" key.

There are keys that automatically interact with the OSDKeyboard 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 32.

When the user presses and releases any key, the code execution jumps to the RemoteKeyPress event (if available). Then, if the key stroke 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 tothe . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

Table 32: Hardcoded Keys in OSDKeyboardComponent

Key	Key Code (Decimal)	Function
Left Arrow	37	Move cursor to next key on left
Right Arrow	39	Move cursor to next key on right
Up Arrow	38	Move cursor to next key above
Down Arrow	40	Move cursor to next key below
Spacebar	32	"Press" virtual key currently highlighted by cursor

3.3.10.6 Creating a Keyboard Template

A keyboard template is created in order to define the look of an onscreen keyboard. A picture can be imported into the template, and then its keys defined, for example, the movement through the keyboard, what keystroke is sent when a particular key is selected, and so on. Once this is done, the keyboard template can be used in the *OSDKeyboard* component.

Follow these steps to create a keyboard template and then a keyboard.

- Select Project Explorer, right click, and select New Item → New keyboard template (see Figure 17).

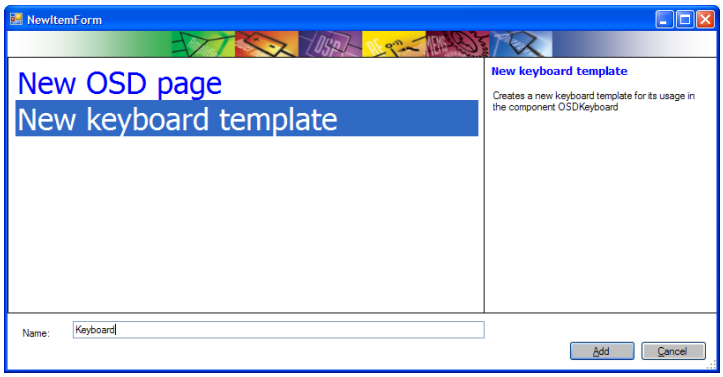


Figure 17: Creating Keyboard Template

- Follow this step to select one background image and one foreground image to define the look of the virtual keyboard.
- Select Property Navigator, BgImage and FgImage for a background image and a foreground image respectively.

  The background image on the left is the keyboard when no key is selected; the foreground image on the right shows how the keyboard will look when a key is selected (see Figure 18). When a key is selected, the foreground picture is overlaid on the background image.

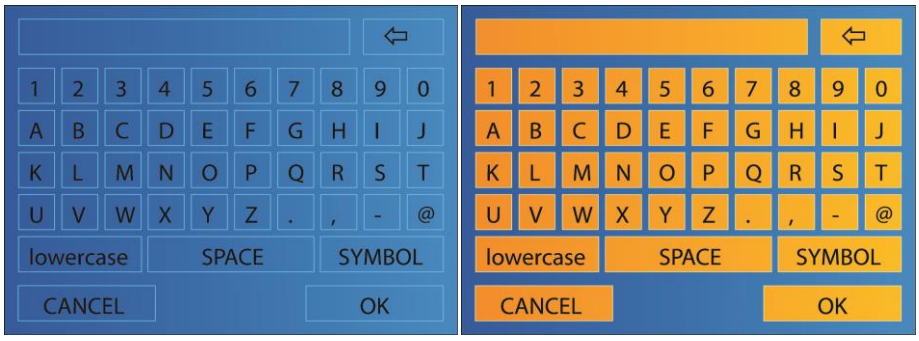


Figure 18: Background and Foreground Keyboard Images

- *OSDKeyboardKey*is used to define the area where the keys are located within the image. This special component allows you to define the movement through the keyboard and the keycode which will be sent when the selected key is pressed.

Drag and drop the component on the image to place it over the keyboard; then, rescale the selection box to match the size of the key.

Define the behavior of the key using the properties shown in the Property Navigator. These are listed in Table 33.

Table 33: OSDKeyboardKey Properties

Property	Short Description
DownFlow	Gets or sets what would be next selected key when down key on remote is pressed
Key	Gets or sets character that will be sent when key is pressed
KeyCode	Gets or sets code that will be sent when key is pressed
LeftFlow	Gets or sets what would be next selected key when left key on remote is pressed
RightFlow	Gets or sets what would be next selected key when right key on remote is pressed
UpFlow	Gets or sets what would be next selected key when up key on remote is pressed
Visible	Gets or sets visibility of key

The character map allowed for the *KeyCode* property is Unicode. This means you can define 65,556 different symbols on the keyboard. Note that *Key* (symbol) and *Keycode* (Unicode value of the symbol in decimal) are complementary properties; setting one also sets the other.

- Define fully the keyboard by adding one *OSDKeyboardKey* component to each of the keyboard keys and by setting the behavior of each of these.

Note:
It is possible to select several *OSDKeyboardKey* boxes (using the control key), and copy and paste them as necessary. It is also possible to automatically fill the flow properties of each key, so you do not need to do it one by one. This is done by right clicking on the key and selecting Auto detect flow.

Figure 19 shows an example of how a keyboard template looks as its keys are being assigned.

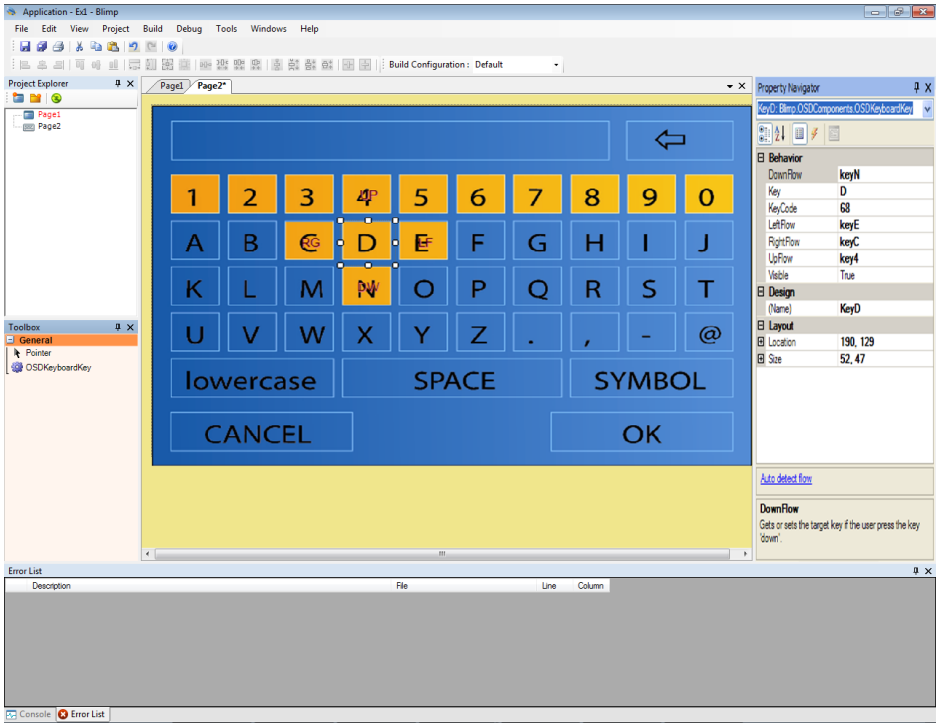


Figure 19: Keyboard Template

- When the keyboard template is complete, follow this step to insert it into the design window

  Place an *OSDKeyboard* component, which will act as the container for the template.

  Note: The *OSDKeyboard* component size should be set to exactly the same size as the template otherwise the keyboard will not be displayed correctly.

  The designed keyboard appears on the list of the template property located on the *Property Navigator*. (It is also possible to change a template during run time).

  Once the designed keyboard is placed on the designer canvas, it looks like the sample in Figure 20.

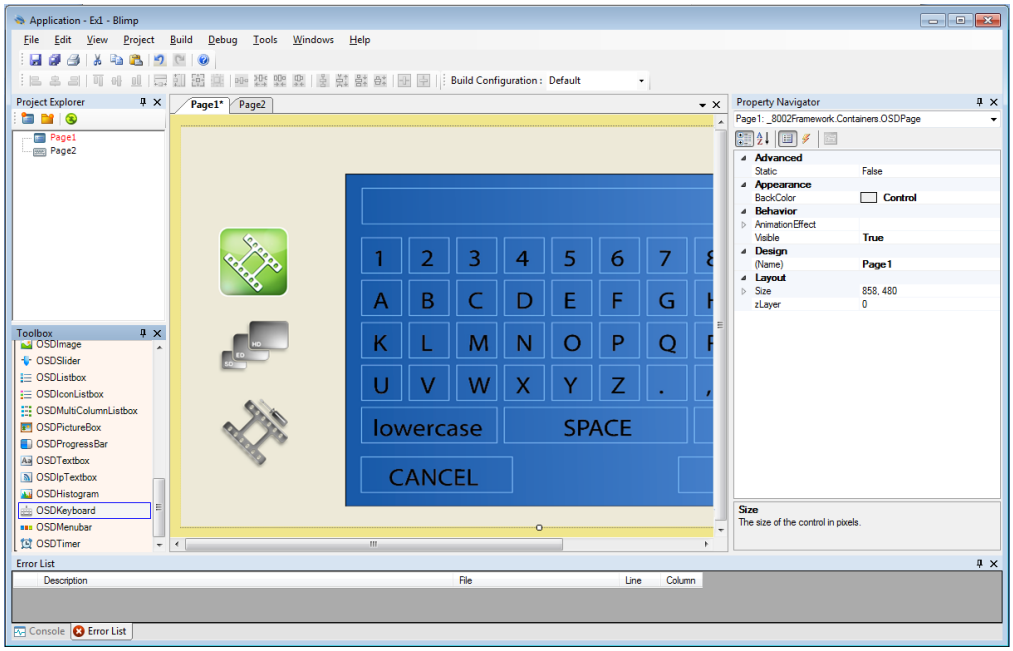


Figure 20: Design Window With OSDMenubar and OSDKeyboard Components Inserted

- *OSDKeyboard* can now be used to input text to an *OSDTextbox*.

3.3.11 OSDHistogram

The *OSDHistogram* allows the displaying of tabulated frequencies shown as bars (bins). It can be used to display preset audio equalizations or to display in real time 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.

This histogram supports up to 256 columns/rows displayed at the same time, and supports minus value.

Table 34: OSDHistogram Properties

Property or Method	Short Description
BinColor	Gets or sets bins color.
MaxValue	Gets or sets the maximum value for the range of each bar.
MinValue	Gets or sets the minimum value for the range of each bar.
TipColor	Gets or sets the Bin tip color.
TipThickness	Gets or sets the Bin tip thickness.
Orientation	Gets or sets orientation (vertical or horizontal) of component .
Spacing	Gets or sets spacing (in pixels) between bars.

TotalBins	Gets or sets total number of bars in which histogram is divided.
setBinValue()	Sets value for each individual bin.
setBinsValue()	sets the value for multiple bins as specified by beginning and end index.

3.3.11.1 BinColor

Gets or sets the bins color.

Property declaration:

```
public uint BinColor {get; set;}
```

Code window usage example:

```
osdHistogram1.BinColor = 0xFFFFFFFF;
```

3.3.11.2 MaxValue

Gets or sets the maximum value for the range of each bar.

Property declaration:

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

Code window usage example:

```
osdHistogram1.MaxValue  = 100;
```

3.3.11.3 MinValue

Gets or sets the minimum value for the range of each bar.

Property declaration:

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

Code window usage example:

```
osdHistogram1.MinValue  = -100;
```

3.3.11.4 Orientation

Gets or sets the orientation (vertical or horizontal) of the component.

Property declaration:

```
public OSD_ORIENTATION Orientation {get; set;}
```

Code window usage example:

```
osdHistogram1.Orientation = OSD_ORIENTATION.HORIZONTAL;
```

3.3.11.5 Spacing

Gets or sets the spacing (in pixels) between the bars.

Property declaration:

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

Code window usage example:

```
osdHistogram1.Spacing = 10;
```

3.3.11.6 TotalBins

Gets or sets the total number of bars in which the histogram is divided. Maximum value is 255 bars.

Property declaration:

```
publicbyte TotalBins {get; set;}
```

Code window usage example:

```
osdHistogram1.TotalBins = 16;
```

3.3.11.7 setBinValue

This method sets the value for each individual bin. 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.

Method declaration:

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

Code window usage example:

```
int i;

osdHistogram1.TotalBins = 30;
osdHistogram1.MaxValue = 30;
osdHistogram1.MinValue = 0;
osdHistogram1.Spacing = 1;

for (i = 0; i < 30; i++)
{
```

```
    osdHistogram1.setBinValue(i, (short)i);
}
```

3.3.11.8 setBinsValue

sets the value for multiple bins as specified by beginning and endindex.

Method declaration:

```
UINT32 setBinsValue(UINT8 valueArraySize, 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);
```

3.3.12 ExternalOSD

The external OSD component provide the capability to display video from an external input, usually from external OSD port onto the Blimp OSD as a sub-window or blended video.

The external OSD resolution should be less than OSD resolution.

The external OSD component can be placed at any location within the Blimp OSD page.

The display area allows to crop and display only an area of the external OSD. The external OSD portion outside the display area will be transparent. See Figure 21.

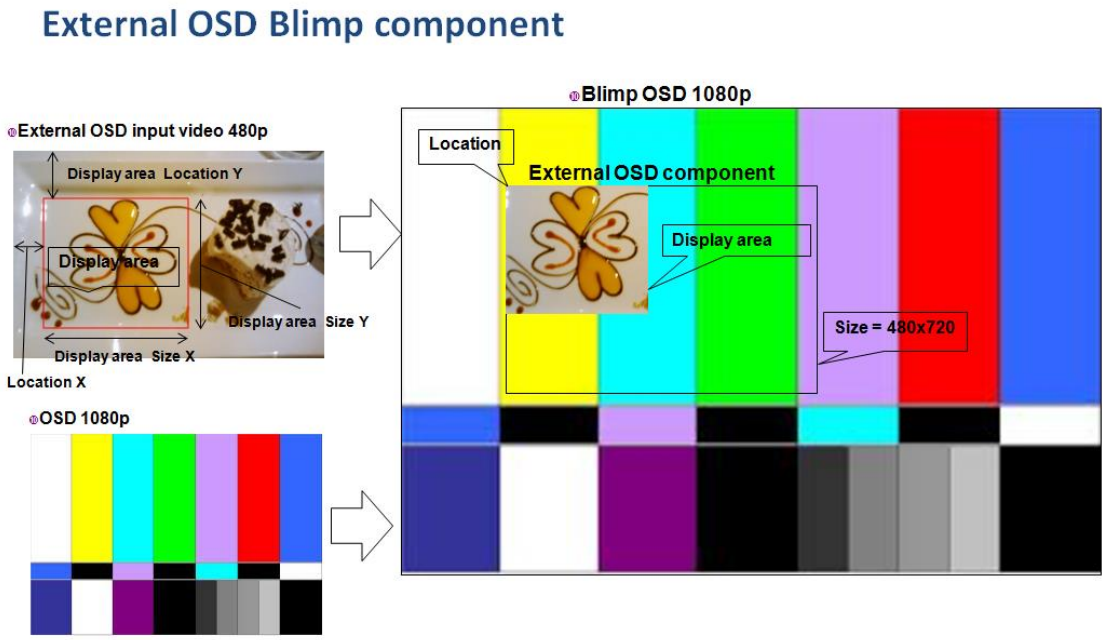


Figure 21: External OSD component display behavior

Table 35: ExternalOSD Properties

Property or Method	Short description
DisplayArea	Sets the window area that will be displayed from incoming externalOSD video.
Source	Selects the video input 1 or 2 to be displayed.
Resolution	Blimp generate code shall use only x and y of Resolution to set size like other components.
ColorSpace	Sets the external OSD video input color space.
Format	Sets the format of the external OSD video input.
setResolution	Defines the video format of external OSD input.

3.3.12.1 DisplayArea

Sets the window area that will be displayed from incoming externalOSD video. The Display Area has a Location and size member. The DisplayArea location specifies the coordinates of the upper-left corner of input video that will be displayed at component location. The DisplayArea size specifies the size of the display area in pixels.

Property Declaration:

```
public Point DisplayLocation {get; set;}
publicSizeDisplaySize {get; set;}

publicvoid setDisplayArea(UINT16 x, UINT16 y, UINT16 w, UINT16 h)
```

Code window usage example:

```
externalOSD1.setDisplayArea(0,0,200,200);
```

3.3.12.2 Source

Selects the video input 1 or 2 to be displayed. Source 2 is only available on ADV8003.

Set the value 0 for External OSD TTL and 1 forVideo TTL.

Property Declaration:

```
public ByteSource {get; set;}
```

Code window usage example:

```
externalOSD1.Source = 0;
```

3.3.12.3 Resolution

Blimp generate code shall use only x and y of Resolution to set size like other components. The Size value will be selectable from the following list:

[480p, 720p, {selected custom value}, customize…]

When selecting customize, a pop will appear to enter width and height of the component .The {selected custom value} will only be displayed if a customize value has been entered.

Property Declaration:

```
public Resolution Resolution {get; set;}
```

3.3.12.4 ColorSpace

Sets the external OSD video input color space.

Set the value 0 for RGB and 1 for YbCr.

Property Declaration:

```
publicByteColorSpace {get; set;}
```

Code window usage example:

```
externalOSD1.ColorSpace = 0;
```

3.3.12.5 Format

Sets the format of the external OSD video input and the range is valid from 0 to 20.

Property Declaration:

```
public ByteFormat {get; set;}
```

Code window usage example:

```
externalOSD1.Format = 0;
```

3.3.12.6 setResolution

Defines the video format of external OSD input.

Method Declaration:

```
publicvoid setResolution(UINT16 width, UINT16 height);
```

Code window usage example:

```
externalOSD1.setResolution(1280,720);
```

3.3.13 OSDLine

This component draws a line between 2 points with specified color and width. It cannot have focus set to it and it does not have any event associated to it.

The line component differs from other component since it uses the MCU processor to draw a vectored line in DDR2 memory space. The line is only drawn once into DDR2 memory during initialization.

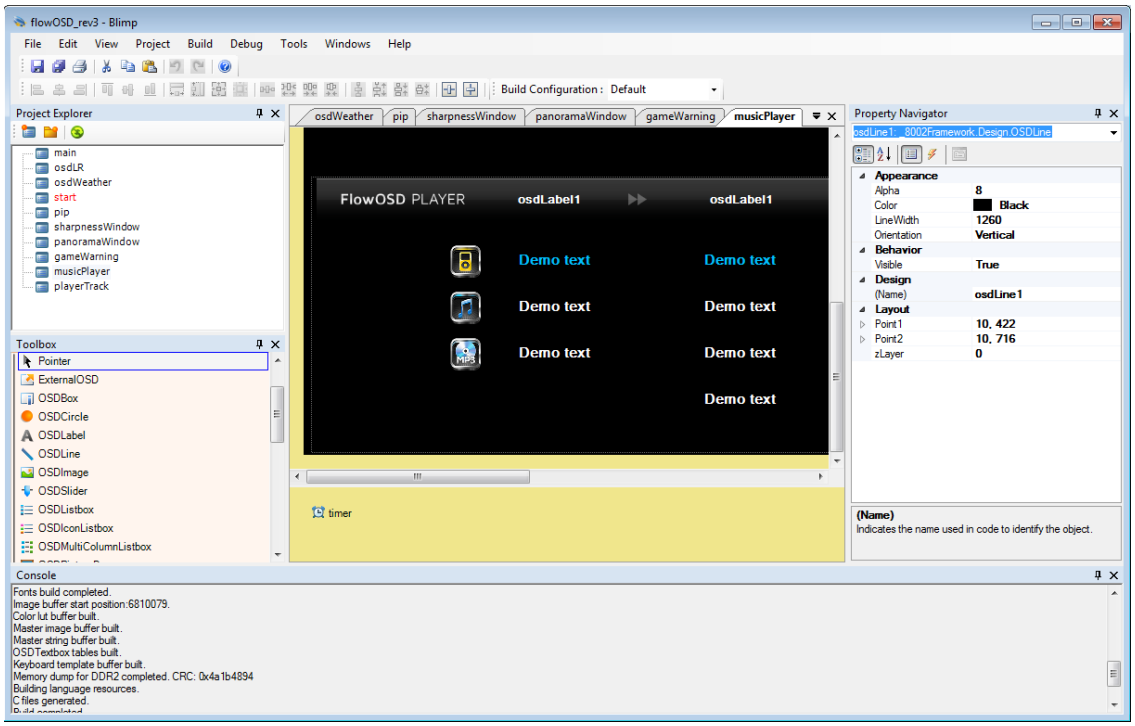


Figure 22: Design Window for OSD Line component (OsdLine overlapped with other components with alpha value 8)

Table 36: OSDLine Properties

Property or Method	Short description
Color	Sets the line color.
LineWidth	Sets the width of the line in pixels.
Orientation	Sets if the line is horizontal, vertical or diagonal.
Point1	Sets the coordinates where the line will start.
Point2	Sets the coordinates where the line will end.

3.3.13.1 Color

Get or Sets the color of the line.

Property Declaration:

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

Code window usage example:

```
//White color
 osdLine1.Color = 0xFFFFFFFFu;
```

3.3.13.2 LineWidth

Sets the width of the line in pixels.

Property Declaration:

```
public ushort LineWidth {get; set;}
```

3.3.13.3 Orientation

Sets the line is horizontal or vertical or diagonal.

Property Declaration:

```
publicOrientationType Orientation {get; set;}
```

3.3.13.4 Point1

Sets the coordinates where the line will start. Range [0 – 4095]

Property Declaration:

```
public Point Point1 {get; set;}
```

3.3.13.5 Point2

Sets the coordinates where the line will end.Range [0 – 4095]

Property Declaration:

```
public Point Point2 {get; set;}
```

3.3.14 OSDPicturebox

A picture box shall allow to draw a box frame which is defined by a reference image with borders and copy the middle sections ddr2 regions to draw any size of a frame box. It cannot have focus set to it and it does not have any event associated to it.

Blimp will tile the picture equally into 9 parts and store each tile in DDR2 memory.

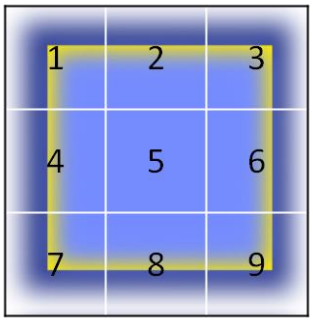


Figure 23: Picture and regions stored in DDR2

Each numbered box will be stored as a region in DDR2 memory.

When drawing different sizes, the corner regions 1, 3, 7 and 9 will always be drawn in the corners. Middle regions will be repeated as required if box size is bigger than original *OSDPicturebox* image.

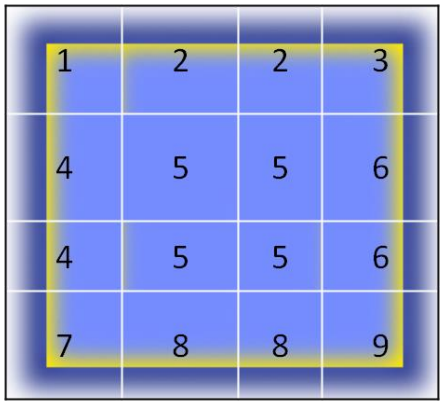


Figure 24: ADV8002 regions for larger box with reference to DDR2 stored image from previous Figure

Note that OSDPicturebox will support maximum regions of 50. When it exceeds the compilation error will be thrown in console window.

Table 37: OSDPicturebox Properties

Property or Method	Short description
ColorDepth	Sets the color format.
Image	Loads the Image for *OSDPicturebox*.
MaxSize	Sets the maximum size of the *OSDPicturebox*to allocate. If 0, the

	specifiedcomponent size will define the maximum. Default is 0.

3.3.14.1 Image

Set the reference image for the PictureBox. The image will be divided into 9 equal tiles.

A popup associated with image property willshows the reference image with clear division lines for 9 areas (see Figure 25) and also the image property will display the information about reference image.

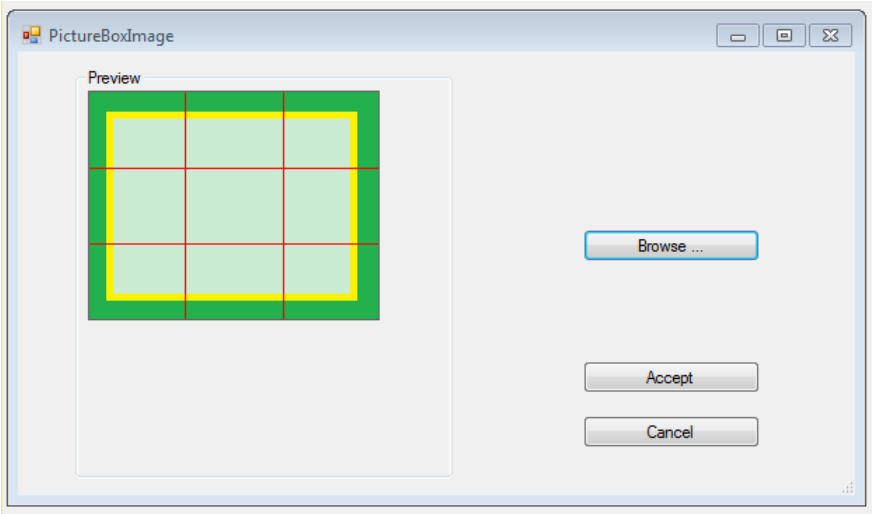


Figure 25: PictureBoxImage window with clear division lines for 9 areas.

3.3.14.2 MaxSize

Sets the maximum size of the picturebox to allocate ddr2 memory. If the value is 0, the specified component size will define as maximum size. Default is 0. Range [0 – 4095]

Property declaration:

```
public Size MaxSize{get; set;}
```

3.3.15 OSDSlider

A slider shall allow to draw the BarImage with the reference image by copying the middle sections and has the single CarretImage that can be moved using the arrow keys.

The *OSDSlider* will support maximum regions of 50. When it exceeds the compilation error will be thrown in console window.

Table 38: OSDSlider Properties

Property or Method	Short Description
BarImage	Sets the image for the bar.
CarretImage	Setsthe image for the carret.
BarAlpha	Allows to control the bar alpha value separately. Note that this alpha value will be normalized to the component alpha value.
CarretAlpha	Allows to control the carret alpha value separately. Note that this alpha value will be normalized to the component alpha value.
ColorDepth	Color format of the icon images.
Orientation	Sets progress bar filling direction (horizontal or vertical).
EndPadding	Sets the minimum and maximum position of the carret on the bar. Default value 0 sets the middle of image on end of bar image.
CarretOffset	Sets the position offset of the carret from center line of the bar. Default value 0 sets center of carret to center of bar.
MinValue	Sets the lower bound value.
MaxValue	Sets the upper bound value.
Step	Sets the value which the progress bar will increment and decrement.
Position	Sets the initial value of the progress bar. This value must be between MinValue and MaxValue
addStep()	Adds the amount defined in the property Step to the current position.
subtractStep()	Subtracts the amount defined in the property Step to the current position.

Table 39: OSDSlider Events

Event	Short Description
RemoteKeyPress	Occurs in response to a user keystroke input
ValueChanged	Occurs whenever the length of the bar gets modified.

3.3.15.1 BarImage

Sets the image for the bar.

Property declaration:

```
public SliderBarImage BarImage{get: set;}
```

The BarImage shall be divided into 3 sections: lower extremity, middle section and higher extremity.

If component is longer than bar image, middle section shall be copied as many time as required to fill area between lower and higher extremities.If component is shorter than bar image, lower and higher extremities will be cropped from side adjacent to middle section.

A popup associated with BarImage property shall allow user to view reference image with clear division lines for 3areas(see Figure 26).

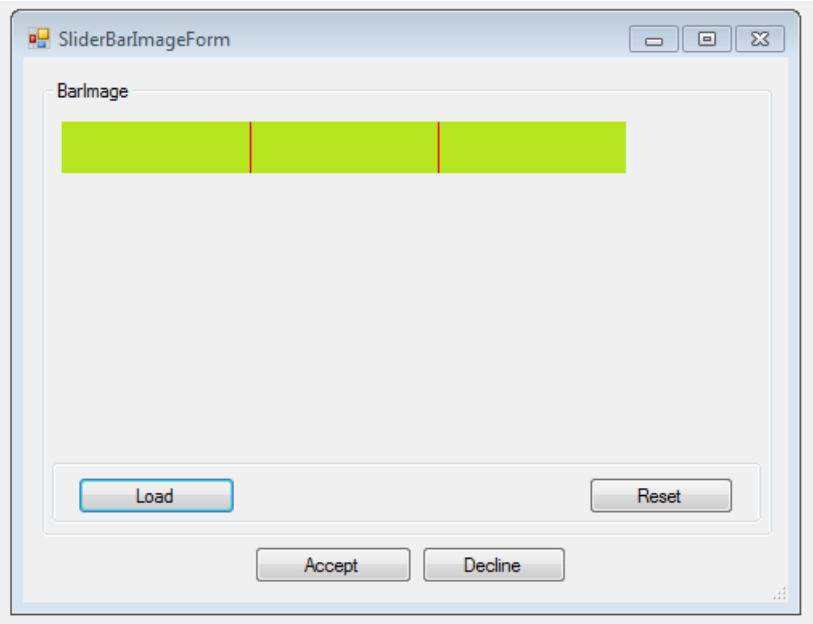


Figure 26BarImage window with clear division lines for 3 areas.

3.3.15.2 CarretImage

Sets the image for the carret.

Property declaration:

```
public Bitmap CaretImage{get; set;}
```

3.3.15.3 BarAlpha

Allows to control the bar alpha value seperately. Note that this alpha value will be normalized to the componetn alpha value.

Property declaration:

```
public Byte BarAlpha{get; set;}
```

Code window usage example:

```
osdSlider1.BarAlpha = 8;
```

3.3.15.4 CarretAlpha

Allows to control the carret alpha value seperately. Note that this alpha value will be normalized to the componetn alpha value.

Property declaration:

```
public Byte CaretAlpha{get; set;}
```

Code window usage example:

```
osdSlider1.CaretAlpha = 8;
```

3.3.15.5 Orientation

Sets progress bar filling direction (horizontal or vertical).

Property declaration:

```
public OSD_ORIENTATION Orientation{get; set;}
```

3.3.15.6 EndPadding

Sets the minimum and maximum position of the carret on the bar. Default value 0 sets the middle of image on the bar image.

Property declaration:

```
public SByte EndPadding{get; set;}
```

Code window usage example:

```
osdSlider1.EndPadding = 50;
```

3.3.15.7 CarretOffset

Sets the position offset of the carret from center line of the bar. Default value 0 sets center of carret to center of bar.

Property declaration:

```
public SByte CaretOffset{get; set;}
```

Code window usage example:

```
osdSlider1.CaretOffset= 50;
```

3.3.15.8 MinValue

Gets or sets the lower bound value. This is the minimum value that the bar can reach, even if the SubtractStep() method is being called.

Property declaration:

```
publicshort MinValue{get; set;}
```

Code window usage example:

```
osdSlider1.MinValue = 0;
```

3.3.15.9 Max Value

Gets or 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;}
```

Code window usage example:

```
osdSlider1.MaxValue = 100;
```

3.3.15.10 Step

Gets or sets the step value. This value will be added or subtracted from the carret image location when calling AddStep() or SubtractStep() methods, or when using the hardcoded keys of the component.

Property declaration:

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

Code window usage example:

```
osdSlider1.Step = 2;
```

3.3.15.11 Position

Gets or Sets the initial value of the progress bar. This is useful to set the default value of the carret image bar location, before the user can edit it. This value must be between *MinValue* and *MaxValue*.

Property declaration:

```
public Int32 Position{get; set;}
```

Code window usage example:

```
osdSlider1.Position = 50;
```

3.3.15.12 addStep

Adds the amount defined in the property *Step* to the current value.

Property declaration:

```
public UINT32 addStep()
```

Code window usage example:

```
//add one step to slider bar when keyboard key 1 is pressed

private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{
   if(*keyCode == 49)
   {
      osdSlider1.addStep();
   }
}
```

3.3.15.13 subtractStep

Subtracts the amount defined in the property *Step* to the current value.

Property declaration:

```
public UINT32 subtractStep()
```

Code window usage example:

```
//subtract one step from slider bar when keyboard key 2 is pressed

private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
{
   if(*keyCode == 50)
   {
      osdSlider1.subtractStep();
   }
}
```

3.3.15.14 ValueChanged

This event occurs whenever the location of the value gets modified.

```
Syntax: ValueChanged (void)
```

3.3.15.15 RemoteKeyPress

Syntax: `private void RemoteKeyPress (Byte *keyCode, Boolean *cancel)`

This event occurs in response to a user keystroke input.

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 component does not receive the keystroke (for example, an OSDSlider will not receive the "right" key).

There are keys that automatically interact with the OSDSlider 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 40. 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.

Note that when using the hardcodedkeys,inOSDSliderthe carret image location isadded/subtracted by an amount equal to the unit defined in the *Step* property.

Table 40: Hardcoded Keys in OSDSliderComponent

Key	Key Code (decimal)	Function
Left Arrow	37	CarretImage moves left (only available when in horizontal orientation)
Right Arrow	39	CarretImage moves right(only available when in horizontal orientation)
Up Arrow	38	CarretImage moves up(only available when in vertical orientation)
Down Arrow	40	CarretImage moves down(only available when in vertical orientation)

3.3.16 OsdIconListbox

An item in the list shall be composed on one image (icon) and one text string. Displays a list of different text strings with one icon through which the user can set the focus slot. The color of the text for this component can change depending on the state of the item (default and focus). This can be set from the properties *DefaultTextColor, FocusedTextColor*.

Table 41: OSDIconListbox Properties

Property or Method	Short Description
DefaultTextColor	Gets or sets the default color for the list items text.
FocusedTextColor	Gets or sets the focused text color.
Font	Gets or sets the font.
IconSettings	Sets icon collection.
TextAlign	Gets or sets the alignment of the text.
ItemHeight	Gets or sets the height of each item in the list box in pixels.
CanvasWidth	Sets the width of the canvas area . This property must be used if horizontal scrolling is selected.
FocusIndex	Gets or sets the focus to the specified index.
FocusSlot	Gets or sets the focus to the specified slot.
TextScrollingSpeed	Gets or sets the speed of the text scrolling animation (pixels/ms). 0 to disable.
IconTextSpacing	Gets or sets the spacing between icon and text.
IconSize	Sets the icon size
LeftToRight	Gets or sets the direction of scrolling.
MaxStringLength	Sets the maximum number of characters the Text property can contain.
MaxVisibleItems	Gets or sets the maximum number of items the list will display.
RollBack	Gets or sets whether the list will roll back to the first item on the list once the last one has been reached and vice versa.
ScrollingIntervalTime	Gets or sets the amount of time the control will wait until scrolling is restarted.
ScrollingSpacing	Gets or sets the spacing between the text during scrolling mode. (pixels).
VerScrollingSpeed	Gets or sets the speed of the vertical scrolling animation through items. (ms). 0 to disable. Range[50-5000mS].
TotalItems	Gets or sets the maximum number of items that the list can contain.
VisibleItems	Gets or sets the number of items that list will display.

Table 42: OSDIconListbox Events

Event	Short Description
RemoteKeyPress	Occurs in response to a user keystroke input.
FocusedItemChanged	Occurs when focus of the item is changed.
VertScrollingFinish	Occurs when the vertical scrolling between icon list items has been completed.

3.3.16.1 DefaultTextColor

Gets or sets the default color for the list items text. This is usually set through the GUI although it may be changed at run time.

Property declaration:

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

Code window usage example:

```
//Set default text to Green color
osdIconListbox1.DefaultTextColor = 0xFF00FF00u;
```

3.3.16.2 FocusedTextColor

Gets or sets the focused text color associated with this component. This is usually set through the GUI although it may be changed at run time.

Property declaration:

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

Code window usage example:

```
//Set focused text to Yellow color
osdIconListbox1.FocusedTextColor = 0xFFFFFF00u;
```

3.3.16.3 Font

Gets or sets the font. For more details please refer the section Selecting fonts.

Code window usage example:

```
osdIconListbox1.Font = new Font("Tahoma", 15f, FontStyle.Regular);
//It is also possible to assign fonts between two components
osdIconListbox1.Font = osdIconListbox2.Font;
```

3.3.16.4 IconSettings

Sets icon collection. Select Icon Settings to access the Icon collection editor, as shown in Figure 27.
All item image within the same list shall be the same size.The size of the image shall be specified by the icon size property of the component.If an image is loaded is smaller, the image will be top-left aligned and remaining

area will be transparent and If an image is loaded is bigger, the image shall be cropped to image size defined in component.

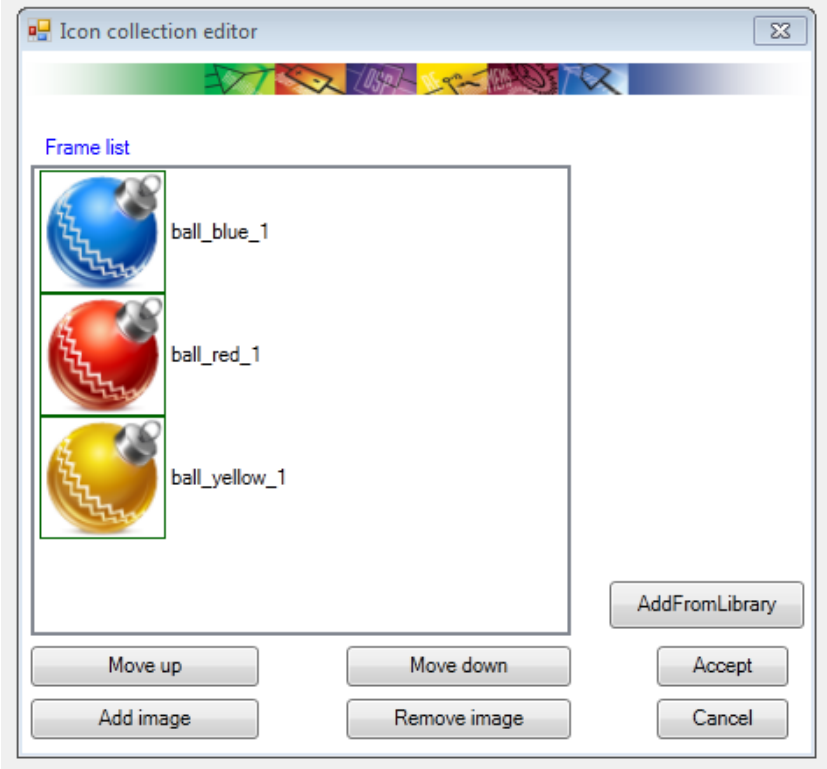


Figure 27: Icon collection editor window

The AddFromLibrary button opens the Image Library GUI window. The Image Library GUI window will collect the entire image which is already used in the blimp project. The user can select an image from library or they can add a new image.

The Icon will be set to the corresponding item text using the setItemIcon Api.

Method declaration:

```
publicsetItemIcon(UINT8 itemIndex, UINT8 iconIndex)
```

Code window usage:

```
// Sets the icon of item index 0, 1for the osdIconListbox1 component
osdIconListbox1.setItemIcon(0,0);
osdIconListbox1.setItemIcon(1,1);
```

3.3.16.5 ItemHeight

Gets or 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.

Property declaration:

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

Code window usage example:

```
osdIconListbox1.ItemHeight = 30;
```

3.3.16.6 CanvasWidth

Gets or sets the width of the canvas area . This property must be used if horizontal scrolling is selected.*CanvasWidth* must be big enough to fit the whole text chunk or the animation will jump.

Property declaration:

```
publicUINT16 CanvasWidth{get; set;}
```

3.3.16.7 FocusIndex

Gets or sets the focus to the specified index. Note that by default, the focus item in a list will always be the first element. This is usually set through the GUI although it may be changed on run time.

Property declaration:

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

Code window usage example:

```
osdIconListbox1.FocusIndex = 2;
```

3.3.16.8 FocusSlot

Gets or sets the focus to the specified slot. The highlighted item will always be displayed on the focus slot. Note that by default, the focus slot in a list will always be the first slot.

Property declaration:

```
public Byte FocusSlot{get; set;}
```

Code window usage example:

```
osdIconListbox1. FocusSlot = 2;
```

The “focus slot” property means which slot will be used as focus position, like:

Visible Item 0	(slot 0)
Visible Item 1	(slot 1)
Visible Item 2	(slot 2)
Visible Item 3	(slot 3)
Visible Item 4	(slot 4)
Visible Item 5	(slot 5)

Code window usage example:

```
//Code used in Iconlistboxes shown in below figures
 public void Load()
 {

      osdIconListbox1.TotalItems = 5;
      osdIconListbox1.VisibleItems = 3;

//setting default color as blue
      osdIconListbox1.DefaultTextColor= 0xff0000ffu;

      //setting focused color as red
      osdIconListbox1.FocusedTextColor = 0xffff0000u;

      osdIconListbox1.setItemIcon(0,0);
      osdIconListbox1.setItemIcon(1,1);
      osdIconListbox1.setItemIcon(2,2);
      osdIconListbox1.setItemIcon(3,3);
      osdIconListbox1.setItemIcon(4,4);
      osdIconListbox1.setIconSize(32,32);
      osdIconListbox1.ItemText[0] = "HDMI 1";
      osdIconListbox1.ItemText[1] = "HDMI 2";
      osdIconListbox1.ItemText[2] = "HDMI 3";
      osdIconListbox1.ItemText[3] = "HDMI 4";
      osdIconListbox1.ItemText[4] = "HDMI 5";
      osdIconListbox1.FocusSlot = 1;//Focus slot1
      OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox1);

}
```

Figure 28: Focus slot 1 preview

The slot 1 has been set to focus slot. The focused item shall change with the up and down arrow keys. An item that is null shall represent an empty item.

3.3.16.9 TextScrollingSpeed

Gets or sets the speed of the text scrolling (pixels/seconds). 0 to disable.Range[0 – 2000]

Property declaration:

```
publicUINT16 TextScrollingSpeed{get; set;}
```

Code window usage example:

```
osdIconListbox1.HorScrollingSpeed = 20;
```

3.3.16.10 VerScrollingSpeed

Gets or sets the speed of the vertical scrolling animation through items (ms). A value of '0' means vertical scrolling disabled. Range[50-5000mS].

Property declaration:

```
publicUINT32 VerScrollingSpeed{get; set;}
```

Code window usage example:

```
osdIconListbox1.VerScrollingSpeed = 20;
```

3.3.16.11 IconTextSpacing

Gets or sets the spacing between icon and text of the *OsdIconListbox*component.

Property declaration:

```
public Byte IconTextSpacing{get; set;}
```

Code window usage example:

```
osdIconListbox1.IconTextSpacing = 10;
```

3.3.16.12 IconSize

Sets the icon size of the *OsdIconListbox*component..If an image is loaded is smaller than the icon size property, the image will be top-left aligned and remaining area will be transparent and If an image is loaded is bigger than the icon size property, the image shall be cropped to icon size defined in component.

Property declaration:

```
public Size IconSize{set;}
```

3.3.16.13 LeftToRight

Gets or sets the direction of scrolling. If set to true, the text scrolls from left to right; if set to false, scrolls from right to left..

Property declaration:

```
publicBoolLeftToRight{get; set;}
```

Code window usage example:

```
osdIconListbox1.LeftToRight = true;
```

3.3.16.14 MaxStringLength

Sets the maximum number of characters the Text property can contain for each item text.

Property declaration:

```
public UINT16 MaxStringLength{set;}
```

3.3.16.15 MaxVisibleItems

Gets or sets the maximum number of items to allocate ddr2 memmory. Note that *MacVisibleItems* must always be bigger than or equal to the *VisibleItems* property.

Property declaration:

```
public Byte MaxVisibleItems{get; set;}
```

Code window usage example:

```
osdIconListbox1.MaxVisibleItems = 3;
```

3.3.16.16 RollBack

Gets or sets whether the list will roll back to the first item on the list once the last one has been reached and vice versa.This is usually set once on the GUI and not changed during run time.

Property declaration:

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

3.3.16.17 ScrollingIntervalTime

Gets or sets the amount of time the control waits until scrolling is restarted. The ScollingIntervalTime is defined in frames. This is usually set once on the GUI and not changed during run time.

Property declaration:

```
publicUINT16 ScrollingIntervalTime{get; set;}
```

3.3.16.18 ScrollingSpacing

Gets or sets the spacing between the text during scrolling mode. (pixels).

Property declaration:

```
publicUINT16 ScrollingSpacing{get; set;}
```

3.3.16.19 TotalItems

Gets or sets the 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:

```
osdIconListbox1.TotalItems = 10;
```

3.3.16.20 VisibleItems

Gets or sets the number of items that list will display in the *OsdIconListbox*, which must be less than or equal to*TotalItems*. If *VisibleItems*<*TotalItems*, the *OsdIconListbox* can be scrolled.

Property declaration:

```
publicbyte VisibleItems{get; set;}
```

Code window usage example:

```
osdIconListbox1.VisibleItems = 2;
```

3.3.16.21 ItemText

Sets an item text as a null terminated ascii string from a buffer of type char[] in the *OSDIconListbox* component. Note that this property is limited to ASCII characters.

Method declaration:

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

Code window usage example:

```
osdIconListbox1.ItemText[0] = "First item of the list";
```

3.3.16.22 ItemTextW

Sets an item text within the *OSDIconListbox item* as a unicode string from a buffer of type uint16[].

Method declaration:

```
public ushort[] ItemTextW{get; set;}
```

Code window usage example:

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

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox1);


    osdIconListbox1.ItemText[0] = "First Item";
    osdIconListbox1.ItemText[1] = "Second Item";
    osdIconListbox1.ItemText[2] = "Third Item";

    //Read 11 characters from osdListbox1 and put the read string into
buffer
    osdIconListbox1.readItemTextW(0,buffer,11);

    //osdListbox1.ItemTextW[1] reads now "First Item"
    osdIconListbox1.ItemTextW[1] = buffer;
}
```

3.3.16.23 ItemConsText

Gets or sets an 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 *LanguageSettings* menu in Blimp.

Method declaration:

```
public UINT32[] ItemConstText
```

Code window usage example:

```
osdIconListbox1.ItemConstText[1] = StringManager.AUDIO_SETTINGS;
//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
```

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

3.3.16.24 readItemTextW

Gets the text of any *OSDIconListbox* item and stores it into a buffer. The read text can be any Unicode character.

Method declaration:

```
publicvoid readItemTextW(ushort* buffer, ushort size);
```

buffer: Array where the read characters are going to be stored in. It should be long enough to hold the value set in *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) which are going to be read from the item. For example, if the item string length is 11, size should be set to 12.

Code window usage example:

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

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox1);

    osdIconListbox1.ItemText[0] = "First Item";
    osdIconListbox1.ItemText[1] = "Second Item";
    osdIconListbox1.ItemText[2] = "Third Item";

    //Read 11 characters from osdListbox1 and put the read string into
```

```
buffer
    osdIconListbox1.readItemTextW(0,buffer,11);

    //osdListbox1.ItemTextW[1] reads now "First Item"
    osdIconListbox1.ItemTextW[1] = buffer;
}
```

3.3.16.25 setItemTextFormat

Sets an item text as 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:

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

Parameters:

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 osdIconListbox
Texts
byte i = 0;

public void Load()
{
for(i=0;i<3;i++)
    {
    osdIconListbox1.setItemTextFormat(i,"HDMI Input %d",i);
```

```
        }
      OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox1);

  }
```

3.3.16.26 setItemIcon

Sets the icon for the specified item index from the icon list collection in the *OSDIconListbox*.

Method declaration:

```
public UINT32 setItemIcon(byte itemindex, byte iconindex);
```

Parameters:

itemindex: Index of the item within the list.

Iconindex: Index of the icon within the icon list collection window. Note: the first icon in the icon window is 0.

Code window usage example:

```
public void Load()
{
osdIconListbox1.setItemIcon(0,0);
    osdIconListbox1.setItemIcon(1,1);
    osdIconListbox1.setItemIcon(2,2);
   OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox1);
  }
```

3.3.16.27 setIconSize

Sets the icon size of the icon listbox.

Method declaration:

```
public UINT32 setIconSize(UINT8 width, UINT8 height);
width , height : width and height of the Icon.
```

Code window usage example:

```
osdIconListbox1.setIconSize(32,32);
```

3.3.16.28 FocusedItemChanged

Syntax: `private void FocusedItemChanged (Byte index)`

This event occurs when the focus of the item is changed, that is, when the user moves through the list with the arrow keys.

Parameters:

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

Code window usage example:

```
 public void Load()
 {
osdIconListbox1.ItemText[0] = "HDMI 1";
     osdIconListbox1.ItemText[1] = "HDMI 2";
     osdIconListbox1.ItemText[2] = "HDMI 3";
     osdIconListbox1.ItemText[3] = "HDMI 4";

     OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox1);
 }

private void osdIconListbox1_FocusedItemChanged(Byte index)
{
    osdLabel1.setTextFormat("The Highlighted HDMI Input is %d", index
+ 1);
}
```

3.3.16.29 RemoteKeyPress

Syntax: `private void RemoteKeyPress ( Byte *keyCode, Boolean *cancel)`

This event occurs in response to a user keystroke input.

Parameters:

***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 OSDIconListboxwill not receive the "down" key.

There are keys that automatically interact with the OSDIconListbox 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 below Table 43.

Table 43: Hardcoded Keys in OSDIconListbox Component

Key	Key Code (decimal)	Function
Up Arrow	38	Move up the list
Down Arrow	40	Move down the list

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.

Code window usage example:

```
   //We could use right and left arrow keys to enable/disable other
submenus
private void osdIconListbox1_RemoteKeyPress(Byte *keyCode, Boolean
*cancel)
{
    if (osdIconListbox1.FocusIndex == 0)
    {
       if (*keyCode == 39) //Right Arrow => Show Submenu
       {
osdIconListbox1.Visible = False;
           osdIconListbox_Submenu.Visible = True;
 OsdApi.ADIAPI_OSDEgSetFocusComponent(osdIconListbox_Submenu);
        }
}
}
```

3.3.16.30 VertScrollingFinish

Syntax: `private void osdIconListbox1_VertScrollingFinish(void)`

This event occurs when the vertical list scrolling animation is completed.

3.3.17 OSDBox

The OSDBox used to draw the box with or without border. It cannot have focus set to it and it does not have any event associated to it.

Table 44OSDBox Properties

Property or Method	Short Description
FillColor	Gets or Sets the box fill color.
BorderColor	Gets or Sets the box border color.
BorderWidth	Sets the border width of the box in pixels.

3.3.17.1 FillColor

Gets or Sets the box fill color in RGB format.

Property declaration:

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

Code window usage example:

```
//Set boxFillColor to Red color
osdBox1.FillColor = 0xffff0000u;
```

3.3.17.2 BorderColor

Gets or Sets the box fill border in RGB format.

Property declaration:

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

Code window usage example:

```
//Set boxBorderColor to Green color

osdBox1.FillColor = 0xff00ff00u;
```

3.3.17.3 BorderWidth

Sets the border width of the box in pixels.

Property declaration:

```
public Byte BorderWidth{get; set;}
```

Code window usage example:

```
osdBox1.BorderWidth = 10;
```

3.3.18 OSDCircle

The OSDCircle is used to draw the circle with different thickness. It cannot have focus set to it and it does not have any event associated to it.

OSDCircle Properties

Property or Method	Short Description
Color	Gets or Sets the circle border color.
BorderWidth	Sets the border width of the circle in pixels

3.3.18.1 Color

Gets or Sets the circle border colorin RGB format.

Property declaration:

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

Code window usage example:

```
//Set Circle BorderColor to Green color

osdCircle1.Color = 0xff00ff00u;
```

3.3.18.2 BorderWidth

Sets the border width of the circle in pixels.

Property declaration:

```
public Byte BorderWidth{get; set;}
```

Code window usage example:

```
osdCircle1.BorderWidth = 10;
```

3.3.19 OSDTimer

The *OSDTimer* implements a timer which calls user defined methods at userdefined intervals. The *OSDTimer* cannot be the target of the page focus, and it defines one event.

OSDTimer accuracy is 500 ms defined for the component.

Table 45OSDTimer Properties

Property or Method	Short Description
Enabled	Gets or sets whether or not timer is running.
Interval	Gets or sets the time, in x500ms units, before *Tick* event gets called.

Table 46OSDTimer Events

Event	Short Description
Tick	Occurs when specified timer interval has elapsed and timer is enabled.

3.3.19.1 Enabled

Gets or sets whether or not the timer is running.

Property declaration:

```
publicbool Enabled {get; set;}
```

3.3.19.2 Interval

Gets or sets the time, in milliseconds, before the *Tick* event gets called in relation to the previous execution time.

Property declaration:

```
publicUNIT32 Interval {get; set;}
```

3.3.19.3 Tick

This event occurs when the specified timer interval has elapsed and the timer is enabled.

Syntax:`privatevoidTick(void)`

Code window usage example:

```
//This example shows a possible use of this component to alternate the
text of two OSDLabel components
bool currentEnabledLabel;
```

```
public void Load()
{
    currentEnabledLabel = false;
    osdTimer1.Interval = 10; //500ms x 10 = 5s
    osdTimer1.Enabled = true;
}
private void osdTimer1_Tick()
{
    currentEnabledLabel = !currentEnabledLabel;
    if (currentEnabledLabel)
    {
        osdLabel1.Text = "Enabled";
        osdLabel2.Text = "Disabled";
    }
    else
    {
        osdLabel1.Text = "Disabled";
        osdLabel2.Text = "Enabled";
    }
}
```

3.3.20 OSDMultiColumnListbox

Displays a list of different multiple column text strings as well as Icons through which the user can move and select None, one or more items may be selected depending on the Selection Mode property. If the number of items exceeds the maximum number of items visible at a time, the OSDMultiColumnListbox develops scrolling functionality. The color of the text for this component can change depending on the state of the item (default, highlighted and selected). This can be set from the properties (DefaultTextColor, HighlightedTextColor, SelectedTextColor, UnselectableColor,Highlighted selectedcolor, Highlighted UnSelectedcolor)in the listbox configuration. User can also link the vertical orientation slider in the multicolumnlistbox for list scrolling functions

.

Table 47Hardcoded Keys in OSDMultiColumnListbox 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.

Table 48OSDMultiColumnListbox Properties

Property or Method	Short Description
TotalItems	Gets or sets the maximum number of items that the list can contain.
VisibleItems	Gets or sets the number of items that list will display..
ItemHeight	Gets or sets the height of each item in the list box in pixels..
Font	Gets or sets the font.
MaxStringLength	Sets the maximum number of characters for all text column the Text property can contain.
IconSettings	Sets the icons for the *OSDMultiColumnListbox*
ListColumns	The column configuration shall be grouped under a sub-group.
ColumnName	User shall be able to modify column name to any name up to 32 characters long.
ColumnType	The type column will allow the user to select whether the column contains text or an image
ColumnWidth	Sets the Column width in pixels.
ColumnAlignment	Sets the alignment of the text or image within the row and column cell.
SelectedColor	Defines the color when an item is the current selection.
Normal Color	Defines the color for an item n the list when not highlighted or selected.
HighlightedColor	Defines the color when an item has cursor over it.
UnSelectableColor	Defines the color for an item in the list when it is not selectable but visible.
HighlightedSelectedColor	Defines the color when an item is the current

	selection and also has cursor over it.
HighlightedUnSelectableColor	Defines the color when an item has cursor over it and item is unselectable
TextAlpha	Sets the alpha value of text for the specified column.
ColumnVisible	Sets if the specified column is visible.
Texts	Texts horizondal scrolling properties shall be grouped under a subgroup.
ScrollPauseTime	Sets the speed in pixels/ms for scrolling if an item text which exceeds the item size. 0 for no scrolling
TextScrollingSpeed	Sets the time between each scrolling and between each item if many text item in one row in seconds
LeftToRight	Sets the direction of the scrolling.
ListScrolling	List scrolling properties shall be grouped under a subgroup.
ScrollingSpeed	Sets the speed of the scrolling (pixels/seconds).
VariableScrolling	Sets the number of clicks that will cause a linear increase in the speed of the listscrolling.
SkipUnselectableItems	Makes list jump over items that are not selectable.
WrapMode	The wrap mode will specify the behavior when a list reaches the first or last item.
Auto Preview	Sets the number of items which will be visible above or below the highlighted item in the list to allow preview of the next item.
FocusSlot	If set, the highlighted item will always be at a fixed position in the list and items will always scroll to show highlighted item in this slot.
Selection mode	Defines how items can be selected in the list.
FocusIndex	If set, the specified index will be focused in the focused slot position.
TextScrollingCloumn	Sets the index of the column that will perform text scrolling.
EnableFocusSlot	If enabled, the highlighted item will always be at focus slot position.
setItemText()	Sets an item text as a null terminated ascii string from a buffer

	of type char[] or unicodes of range 0x80 - 0x7ff or the range 0x800 - 0xffff.
ItemConstText	Sets an text as a string from the multilanguage string table. The StringID can be used to specify the string using StringManager.
setItemTextW()	Sets an item text as a unicode string from a buffer of type uint16[].
readItemTextW()	Read the item text property using the determined formatting settings.
setItemTextFormat()	Sets an item text as a formatted text string in ascii. This method is very similar to the sprint function in C.
setItemIcon()	Sets the icon for the specified item index from the flash.
UnselectableItems	Sets or Gets items that appear unselectable in the list. The color is defined by Unselectable ItemColor. If SkipUnselectableItems flag is set, scrolling through the list will jump over the unselectable items.
HiddenItems	Sets or Gets items that will not appear in the list.
IsScrolling	Returns TRUE is the highlighted item has text scrolling. FALSE otherwise.
OffSetIndex	Gets or sets the offset of the OSDMultiColumnListbox items. It will read the offset between the visible first element of the list and the currently highlighted one. This offset will also take into account hidden items.

Table 49OSDMultiColumnListbox 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.
TextScrollingFinish	Occurs when Text scrolling has been completed.
ListScrollingFinish	Occurs when the List scrolling between list items has been completed.

3.3.20.1 TotalItems

Gets or sets the maximum number of items that the list can contain. Note that *TotalItems* must always be bigger than or equal to the *VisibleItems* property. This is usually set through the GUI although it may be changed on run time.

Property declaration:

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

Code window usage example:

```
osdMultiColumnListbox1.TotalItems = 4;
```

3.3.20.2 VisibleItems

Gets or sets the number of items that list will display, which must be less than or equal to*Total Items*. If *VisibleItems*<*TotalItems*, the MultiColumnlistbox can be scrolled.This is usually set through the GUI and that this value cannot be changed on runtime.

Property declaration:

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

3.3.20.3 ItemHeight

Gets or 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 and that this value cannot be changed on runtime.

Property declaration:

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

3.3.20.4 Font

Gets or sets the font.

For more details please refer the section Selecting fonts.

Code window usage example:

```
osdMultiColumnListbox1.Font = new Font("Tahoma", 15f,
FontStyle.Regular);
//It is also possible to assign fonts between two components
osdMultiColumnListbox1.Font = osdListbox2.Font;
```

3.3.20.5 MaxStringLength

Sets the maximum number of characters for all text columns the Text property can contain.

Property declaration:

```
public UINT16 MaxStringLength{set;}
```

3.3.20.6 IconSettings

Sets the icon collection.Select Icon Settings to access the Icon collection editor, as shown in Figure 29
All item image within the same list shall be the same size.The size of the image shall be specified by Column width and ItemHeight properties of the component.

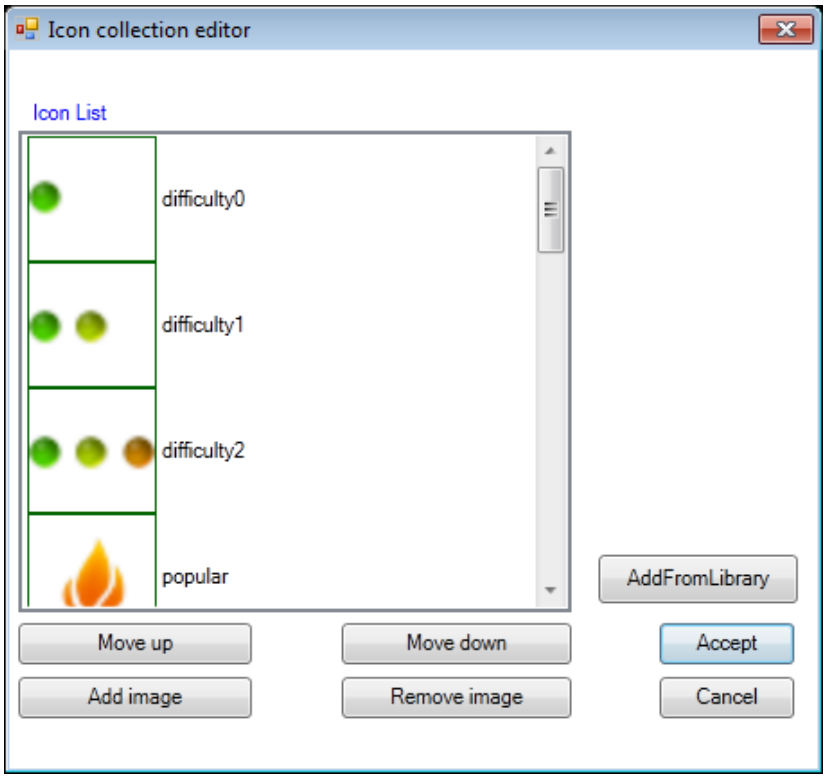


Figure 29: Icon collection editor window

The AddFromLibrary button opens the Image Library GUI window. The Image Library GUI window will collect the entire image which is already used in the blimp project. The user can select an image from library or they can add a new image.

To set icon for the column please see the setItemIcon Api.

3.3.20.7 ListColumns

The column configuration shall be grouped under a sub-group. Select ListColumns to access the Column Configuration window, as shown in Figure 30.

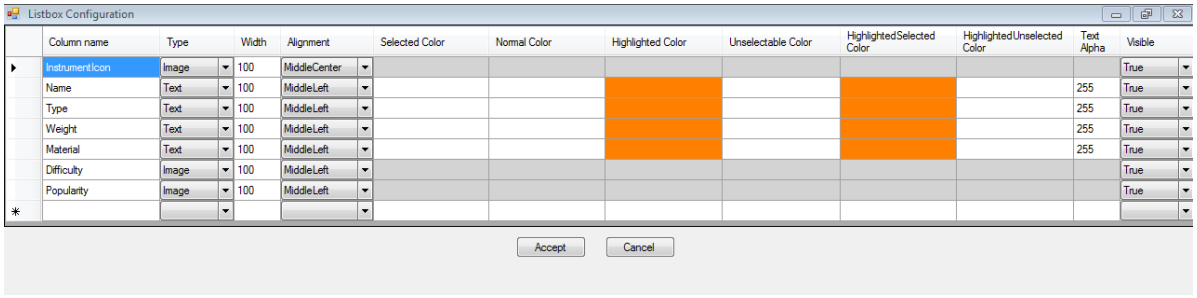


Figure 30:Multi column Listbox configuration

3.3.20.8 ColumnName

User shall be able to modify column name to any name up to 32 characters long.

Property declaration:

```
public stringColumnName{get; set;}
```

3.3.20.9 ColumnType

The type column will allow the user to selecwhether the column contains text or an image.

Property declaration:

```
public boolColumnType{get; set;}
```

3.3.20.10 ColumnWidth

Sets the Column width in pixels.Range[0- 4095]

Property declaration:

```
public Int16ColumnWidth{get; set;}
```

3.3.20.11 ColumnAlignment

Sets the alignment of the text or image within the row and column cell.

Property declaration:

```
public OSD_CONTENTALIGHMENT ColumnAlignment{get; set;}
```

Code window usage example:

```
//sets the Alignment for column index 1.

osdMultiColumnListbox1.Alignment[1] = OSD_CONTENTALIGNMENT.TOPLEFT;
```

3.3.20.12 NormalColor

Defines the color for an item n the list when not highlighted or selected.

Property declaration:

```
public ColorNormalColor{get; set;}
```

Code window usage example:

```
//sets the NormalText color for column index 2.

osdMultiColumnListbox1.setNormalTextColor(2,0xffff0000);
```

3.3.20.13 SelectedColor

Defines the color when an item is the current selection.

Property declaration:

```
public ColorSelectedColor{get; set;}
```

Code window usage example:

```
//sets the SelectedText color for column index 0.

osdMultiColumnListbox1.setSelectedTextColor(0,0xffffff00);
```

3.3.20.14 HighlightedColor

Defines the color when an item has cursor over it.

Property declaration:

```
public ColorHighlightedColor{get; set;}
```

Code window usage example:

```
//sets the HighlightedText color for column index 2.

osdMultiColumnListbox1.setHighlightedTextColor(2,0xffff00ff);
```

3.3.20.15 HighlightedSelectedColor

Defines the color when an item is the current selection and also has cursor over it.

Property declaration:

```
public ColorHighlightedSelectedColor{get; set;}
```

Code window usage example:

```
//sets the HighlightedSelectedText color for column index 0.

osdMultiColumnListbox1.setHighlightSelectedTextColor(0,0xffffff00);
```

3.3.20.16 UnSelectableColor

Defines the color for an item in the list when it is not selectable but visible.

Property declaration:

```
public ColorUnSelectableColor{get; set;}
```

Code window usage example:

```
//sets the UnSelectableText color for column index 0.

osdMultiColumnListbox1.setUnSelectableTextColor(0, 0xffffff00);
```

3.3.20.17 HighlightedUnSelectableColor

Defines the color when an item has cursor over it and item is unselectable.

Property declaration:

```
public ColorSelectedColor{get; set;}
```

Code window usage example:

```
//sets the highlightedUnSelectable color for column index 1.

osdMultiColumnListbox1.setHighlightedUnSelectableTextColor(1,0xffffff00)
;
```

3.3.20.18 TextAlpha

Sets the alpha value of text for the specified column. Note: this property not implemented in current blimp version.

3.3.20.19 ColumnVisible

Sets if the specified column is visible.

Property declaration:

```
public ColorColumnVisible{get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.ColumnVisible[0] = true;
```

3.3.20.20 ScrollingPauseTime

Sets the time between each scrolling and between each item if many text item in one row inseconds

Property declaration:

```
public UInt16ScrollingPauseTime{get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.ScrollingPauseTime= 10;
```

3.3.20.21 TextScrollingSpeed

Sets the speed in pixels/ms for scrolling if an item text which exceeds the item size.0 for no scrolling.

Property declaration:

```
public UInt16TextScrollingSpeed{get; set;}
```

3.3.20.22 LeftToRoght

Gets or sets the direction of scrolling. If set to true, the text scrolls from left to right; if set to false, scrolls from right to left.

Property declaration:

```
publicBoolLeftToRight{get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.LeftToRight = true;
```

3.3.20.23 ScrollingSpeed

Sets the speed of the scrolling (pixels/seconds). Custom values are in seconds and range [0 - 4].This is usually set through the GUI although it may be changed on run time.

Property declaration:

```
public stringScrollingSpeed{get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.ListScrollingSpeed= 2;
```

3.3.20.24 VariableScrolling

Sets variable speed scrolling which increases when multiple click are pressed. It defines the number of clicks that will cause a linear increase in the speed of the listscrolling
and range [0 - 20]

Speed multiplier factor will = number of items left to scroll in queue

For example, when up button pressed 3 times, scrolling item queue is 3, speed will be 3 x scrolling speed. When 1 item is scrolled, the scroll queue becomes 2 and speed is 2 x scrolling speed. When only 1 item left to scroll in queue, speed will be equal to scrolling speed (1x).
This means that every time the scrolling of an item completes, the speed must be updated.
This creates the variable scrolling speed so items are scrolled faster when multiple click are pressed so the item is reached faster.

Property declaration:

```
public byteVariableScrolling{get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.VariableScrolling= 2;
```

3.3.20.25 LinkSlider

Links a OsdSlider component to automatically size to the listbox. The associated OsdSlider will have its position and size override when it is linked to the list.

Property declaration:

```
public objectLinkSlider{get; set;}
```

3.3.20.26 Skip Unselectable Items

Makes the list jump over the items that are not selectable while scrolling list items.

Property declaration:

```
public BoolSkipUnselectableItems {get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.SkipUnselectableItems = true;
```

3.3.20.27 Wrap Mode

The wrap mode will specify the behavior when a list reaches the first or last item. This is usually set through the GUI and that this value cannot be changed on runtime.
Jump - Jump mode causes the highlighted item to jump between first item to last item when reaching beginning or end of list.
Rollback - Roll back mode makes the list continuous with first or last item appearing as the next item in the list when scrolling.

Property declaration:

```
public OSD_WRAPMODE WrapMode {get: set;}
```

3.3.20.28 Auto Preview

Sets the number of items which will be visible above or below the highlighted item in the list to allow preview of the next item.This is usually set through the GUI and that this value cannot be changed on runtime.
Property declaration:

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

If AutoPreview more than number of floor(visible items / 2), the floor(visible items / 2) value will be used.
If totalItems<= Visible Items, AutoPreview will not be active.
When rollback enabled, rollback items will appear within the autopreview.

When jumpmode enabled with no rollback, autopreview will not show rollback items from start or end of list.

3.3.20.29 Focus Slot

If set, the highlighted item will always be at a fixed position in the list and items will always scroll to show highlighted item in this slot.This is usually set through the GUI and that this value cannot be changed on runtime.
Property declaration:

```
public UINT8 FocusSlot {set;}
```

3.3.20.30 Selection Mode

Defines how items can be selected in the list.This is usually set through the GUI and that this value cannot be changed on runtime.

Property declaration:

```
public OSD_MULTICOLUMNLISTBOXMODESelectionMode{set;}
```

```
public enum {
NoSelection,
SingleSelection,
MultipleSelection,
} OSD_MULTICOLUMNLISTBOXMODE;
```

NoSelection – Item selection will be disabled in the display list.

SingleSelection – It enables the single item selection of the display list.

MultipleSelection – It enables the multiple item selection of the display list.

3.3.20.31 Focus Index

If set, the specified index will be focused in the focused slot position.This is usually set through the GUI and that this value cannot be changed on runtime.

Property declaration:

```
public ByteFocusIndex {set;}
```

3.3.20.32 Text Scrolling Column

Sets the index of the column that will perform text scrolling. This is usually set through the GUI although it may be changed on run time.

Property declaration:

```
public ByteTextScrollingColumn {get; set;}
```

Code window usage example:

```
osdMultiColumnListbox1.TextScrollingColumn= 2;
```

3.3.20.33 EnableFocusSlot

If enabled, the highlighted item will always be at focus slot position.

Property declaration:

```
public BoolEnableFocusSlot {set;}
```

3.3.20.34 Set ItemText

Sets an item text as a null terminated ascii string from a buffer of type char[] or unicodes of range 0x80 - 0x7ff or the range 0x800 - 0xffff.

Method declaration:

```
public UInt32SetItemText(UINT8 ColumnIndex, UINT8 Itemindex, String itemText)
```

Code window usage example:

```
osdMultiColumnListbox1.setItemText(0,1,"Itemtext");
```

3.3.20.35 Set ItemTextW

Sets an item text as a unicode string from a buffer of type uint16[]

Method declaration:

```
public UInt32SetItemTextW(UINT8 ColumnIndex, UINT8 Itemindex, UINT8 stringBuffer)
```

Code window usage example:

```
ushort

osdMultiColumnListbox1.setItemTextW(ColumnIndex, Itemindex, buffer);
```

3.3.20.36 Item ConstText

Sets an text as a string from the multilanguage string table. The StringID can be used to specify the string using StringManager.

Method declaration:

```
public UInt32ItemConstText(UINT8 ColumnIndex, UINT8 Itemindex, StringManager id)
```

Code window usage example:

```
osdMultiColumnListbox1.setItemConstText(0, 2, StringManager.AUDIO);
```

3.3.20.37 readItemTextW

Read the item text property using the determined formatting settings.

Method declaration:

```
public UInt32readItemTextW(UINT8 ColumnIndex, UINT8 Itemindex, ushort*
buffer, ushort size)
```

Code window usage example:

```
ushort[] buffer = new ushort[9];
osdMultiColumnListbox1.setItemText(0,1,"Itemtext");

osdMultiColumnListbox1.readItemTextW(0, 1, buffer, 9);

osdLabel1.TextW = buffer;
```

3.3.20.38 SetItemTextFormat

Sets an item text as a formatted text string in ascii. This method is very similar to the sprint function in C. Refer Table 10for list of supported formats.

Method declaration:

```
publicUInt32setItemTextFormat(Columnindex, index, format, paramsobject[]
args)
```

Columnindex: Column index of the *OSDMultiColumnListbox*component

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
OSDMultiColumnListbox(3 columns, 3 items) Texts
```

```
byte i = 0;

public void Load()
{
   for(i=0;i<3;i++)
   {
       osdMultiColumnListbox1.setItemTextFormat(i,i,"HDMI Input %d",i);
   }
   OsdApi.ADIAPI_OSDEgSetFocusComponent(osdMultiColumnListbox1);
}
```

3.3.20.39 SetItemIcon

Sets the icon for the specified item index from the icons added in the icon collection editor.

Method declaration:

```
public UInt32 setItemIcon(byte columnIndex, byte itemindex,

byte iconindex);
```

Code window usage example:

```
// Sets icon for the 0 column of first item.

// Icon index depends upon the icons order of present in icon collection
editor.

osdMultiColumnListbox1.setItemIcon(0, 1, 0);
```

3.3.20.40 UnSelectableItems

Sets or Gets items that appear unselectable in the list. The color is defined by Unselectable ItemColor. If SkipUnselectableItems flag is set, scrolling through the list will jump over the unselectable items.

Property declaration:

```
public Boolean[] UnselectableItems;
```

Code window usage example:

```
/* set all the columns second itemtext as unselectable */
osdMultiColumnListbox1.UnselectableItems[1] = true;
```

3.3.20.41 HiddenItems

Sets or Gets items that will not appear in the list.

Property declaration:

```
public Boolean[] HiddenItems;
```

Code window usage example:

```
/* set all the columns second itemtext and icon as Hidden*/
osdMultiColumnListbox1.HiddenItems[1] = true;
```

3.3.20.42 IsScrolling

Returns TRUE if the highlighted item has text scrolling. FALSE otherwise

Property declaration:

```
Public BooleanIsScrolling;
```

Code window usage example:

```
bool scrolling;

/* Gets the highlighted item scrolling status*/
scrolling = osdMultiColumnListbox1.IsScrolling;
```

3.3.20.43 OffSetIndex

Gets or sets the offset of the OSDMultiColumnListbox items. It will read the offset between the visible first element of the list and the currently highlighted one. This offset will also take into account hidden items.

Property declaration:

```
Public UINT8OffSetIndex;
```

Code window usage example:

```
osdMultiColumnListbox1.OffSetIndex = 2;
```

3.3.20.44 HighlightedItemChanged

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

```
Syntax: HighlightedItemChanged(Byte index)
```

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

Code window usage example:

```
public void Load()
{
    osdMultiColumnListbox1.setItemText(0,0,"HDMI1");
    osdMultiColumnListbox1.setItemText(0,1,"HDMI2");
```

```
    osdMultiColumnListbox1.setItemText(0,2,"HDMI3");
    osdMultiColumnListbox1.setItemText(0,3,"HDMI4");

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdMultiColumnListbox1);
 }

private void osdMultiListbox1_HighlightedItemChanged(Byte index)
{
        osdMultiColumnListbox1osdLabel1.setTextFormat("The Highlighted
HDMI Input is %d", index + 1);
}
```

3.3.20.45 SelectedItemChanged

This event occurs when the selected item is changed, that is, when the user sends the spacebar key.

Syntax: `SelectedItemChanged(Byte index, Boolean newStatus)`

Parameters:

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()
{
    osdMultiColumnListbox1.setItemText(0,0,"HDMI1");
    osdMultiColumnListbox1.setItemText(0,1,"HDMI2");
    osdMultiColumnListbox1.setItemText(0,2,"HDMI3");
    osdMultiColumnListbox1.setItemText(0,3,"HDMI4");

    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdMultiColumnListbox1);
}


private void osdMultiColumnListbox1_SelectedItemChanged(Byte
index,Boolean newStatus)
{
//We get the same result with both instructions. Note however that, if
using SCROLLING_MULTI_SELECTION, //the second implementation would still
work: “index” value would be the last selected item.
```

```
        osdMultiColumnListbox1osdLabel1.setTextFormat("HDMI Input
Selected %d", osdMultiColumnListbox1.SelectedIndex + 1);
        osdMultiColumnListbox1osdLabel1.setTextFormat("HDMI Input
Selected %d", index + 1);
 }
```

3.3.20.46 RemoteKeyPress

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

This event occurs in response to a user keystroke input.

Parameters:

***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 *OSDMultiColumnListbox*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 inTable 13.

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 tothe . See the Pressed Key Event Flow between Pages and Components section on page 21 for more information.

Code window usage example:

```
//We could use right and left arrow keys to enable/disable other
submenus
private void osdMultiColumnListbox1_RemoteKeyPress(Byte *keyCode,
Boolean *cancel)
{
    if (osdMultiColumnListbox1.FocusIndex == 0)
    {
```

```
        if (*keyCode == 39) //Right Arrow => Show Submenu
         {
            osdMultiColumnListbox1.Visible = False;
            osdMultiColumnListbox1_Submenu.Visible = True;
 OsdApi.ADIAPI_OSDEgSetFocusComponent(osdMultiColumnListbox1_Submenu);
        }
}
}
```

3.3.20.47 TextScrollingFinish

This event occurs when the horizontal scrollingof the highlighted itemis completed. Note that the event triggers just at the moment the text stops scrolling, that is, before waiting for the period defined by *ScrollingIntervalTime.*

Syntax: `private void TextScrollingFinish(void)`

3.3.20.48 ListScrollingFinish

This event occurs when the list scrolling finished.

Syntax: `private void ListScrollingFinish(void)`

4 Splash Page

User shall able to design the Splash page in blimpand the page shall be displayed immediately after power-up using HDMI free-run mode. The splash page will not have any user interaction or key press since DMA will be in use for decompression of main binary data and page shall support animation and any font configured.

In Blimp OSD project, the SplashPage folder will be created in project explorer asdefault when creating a new project. User can create a splash page under the “SplashPage” folder as shown in Figure 31. Only one splash page can be supported for now. All resources used in splash page shall be stored in separate binary flash block at the beginning of the flash binary after the header data.

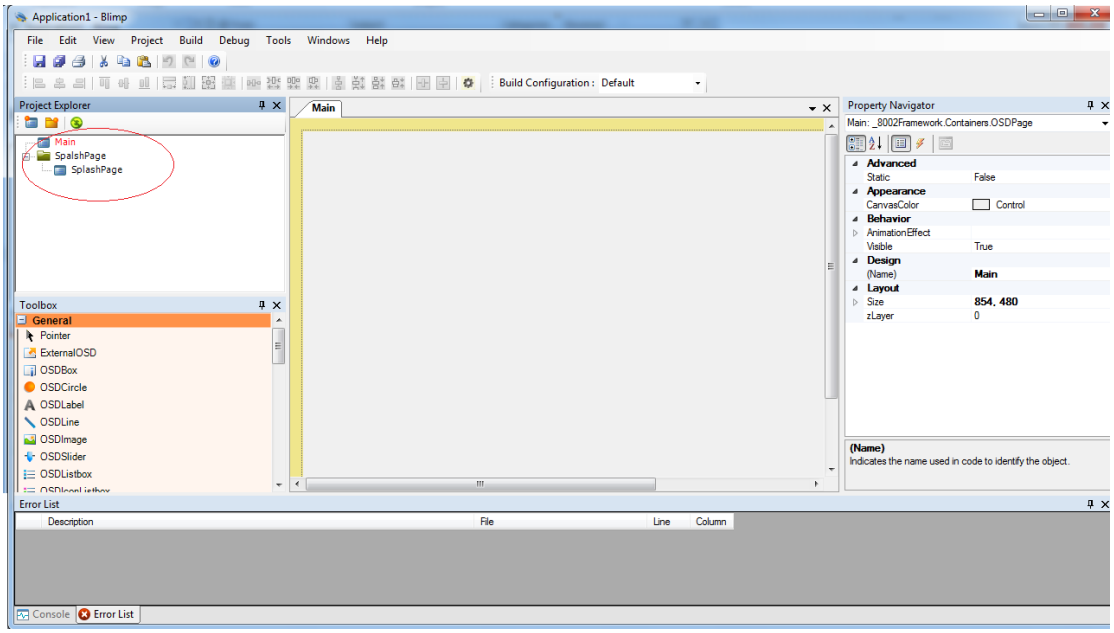


Figure 31:SplashPage folder in Project Explorer window

5 Setting Color Depth

Blimp supports several color depths for the *OSDImage*, *OSDProgressbar*, *OSDMenubar*, *OSDPicturebox*, *OSDSlider* and *OSDKeyboard* components. This allows the user to save OSD storage space and DDR2 bandwidth by selecting the most appropriate bit depth for each component.

Follow these steps to set color depth.

- Select Property Navigator → ColorDepth.
- Set the following color depths, as appropriate:
- TEXT: Color depth (8-bit alpha) used by Blimp to display fonts. Not selectable for graphical components.
- PALETTE: 8-bit Palette color depth support 256 unique colors.
- RGB565: 16-bit color depth (5 bits for red, 6 bits for green, and 5 bits for blue components)
- ARGB4444: 12-bit color depth with alpha channel (4 bits for red, 4 bits for green, 4 bits for blue components, and 4 bits for alpha channel).
- ARGB8888: 24-bit color depth with alpha channel (8 bits for red, 8 bits for green, 8 bits for blue components, and 8 bits for alpha channel).

Note: Blimp displays the images in the same way, that is, in the native format of the picture inserted independently of the color depth selected, although the memory dump and compiled files take it into account.

6 Selecting fonts

Blimp supports Font property for the *OSDLabel*, *OSDListbox*, *OSDTextbox*, *OSDIconListbox*, *OSDMulticolumnListbox* and *OSDIPTextbox* components. We could select the customized font or can assign the multiple fonts through the Font style configuration. It is also 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 which are not allowed in Blimp even though the operating system fully supports them:

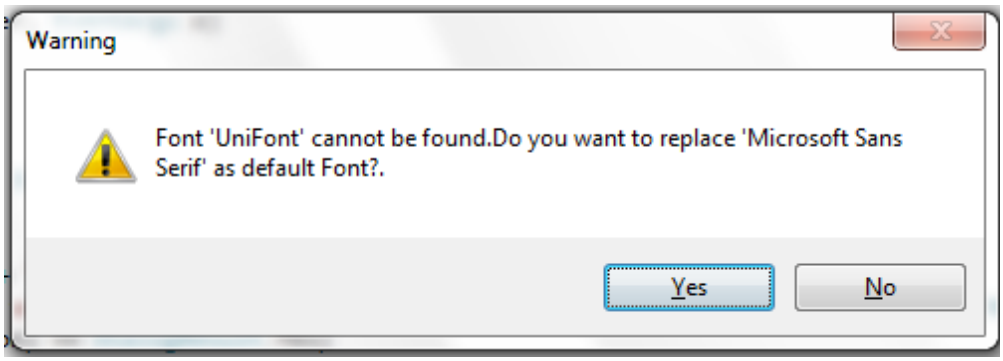
Device-specific/printer fonts

Raster/bitmap fonts
OpenType (erratic; Microsoft fonts are allowed, Adobe fonts are not allowed)
Type 1 fonts

Property declaration:

```
public BlimpFont Font {get; set;}

Code window usage example:
osdLabel1.Font = new Font("Arial", 15f, FontStyle.Regular);
//FontStyle: Regular, Bold, Italic, Underlined, Strikeout

//You can as well assign a current font to a new control
osdLabel1.Font = osdLabel2.Font;
```

Note that if the font is not available in the system while opening the existing project, the warning message will be displayed as shown below figure. If the user select "Yes" in warning message, the font "Microsoft Sans Serif" will be replaced else the project will be closed



7 Getting Max font width

The blimp will have the toolbar option click “Get Width of the Widest Char..." under the project menu and it will be enabled when selecting any text components (OSDLabel, OSDListbox, OSDIconlistbox, OSDMultiColumnListbox, OSDTextbox, &OSDIpTextbox) as shown below.

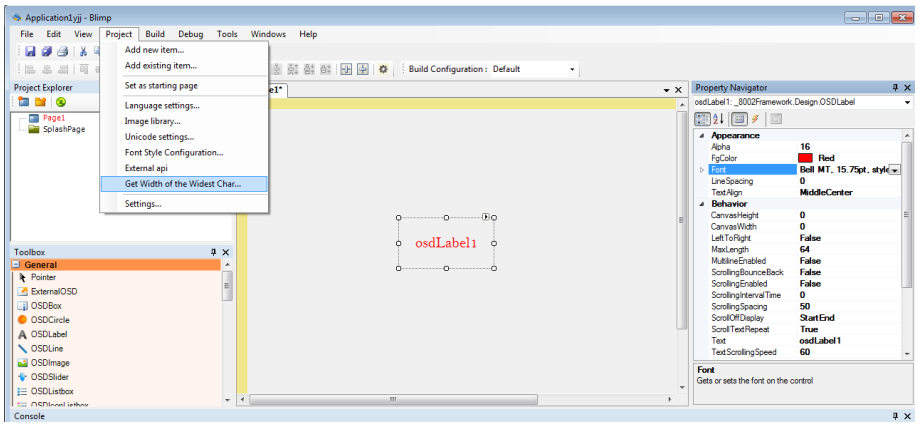


Figure 32:Feature to give width of widest character in font character set

When selecting this option the corresponding Font will be compiled with its selected Unicode characters and the max width of the character in that corresponding font range will be calculated and displayed in GUI using Messagebox. This will be useful to set the component width

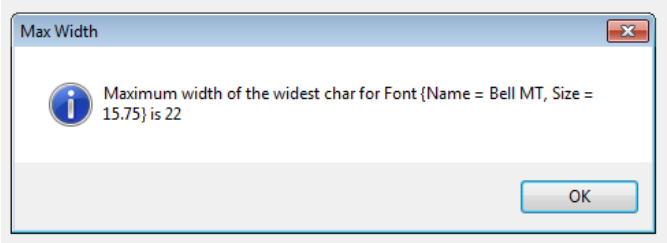


Figure 33:Message box shows the Max char width

8 Blimp Code window preprocessor support

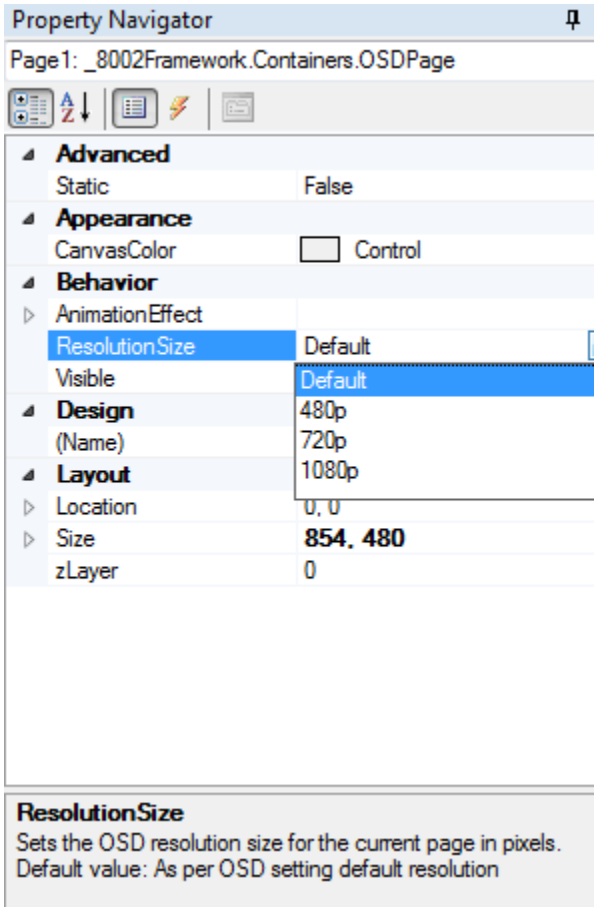
OsdPage and ExternalApi code window should accept preprocessors definitions. Preprocessors #if, #ifdef, #ifndef, #elif, #else, #ifdefined, #define are supported in blimp script window.

Code Window usage Example:

```
#ifdef blimpOsdDebug
 public void EnableDebugLabel()

{

 debugLabel.Visible = true;

}

//write all the functions for Osd debug

#endif

public enum OsdMenu
 {
  InputSettings,
  OutputSettings,
 #ifdefined defined(OSD_MENU_DESIGN1 || OSD_MENU_DESIGN2 )
  AdvanceSettings
 #endif
 }

#if AdvanceSettings
 AdvanceSettingListbox.Visible = true;
 ConfigureAdvanceSettings();
 #endif
```

9 Multi-Resolution OsdPage Configuration

Each page shall have the resolution size property to allow overriding the default resolution. ResolutionSize property of OSD page allows user to configure each page of OSD project configure with different resolution size. Default value sets the default resolution size which is set in the OSD project setting. Predefined ranges are 480p, 720p &1080p. It also accepts custom size value and the value should be in the format of Width, Height.



When Osd page visible property turns from false to true OSD should be scaled depends on the resolution size set for that page.

Overlapping pages of different resolution may give unexpected display results as the last OSD resolution will be configured.Unexpected result means that the system will not care of what will be displayed. Graphics from both pages will likely be scaled one over the other un-proportionally.A log message will be provided when this occurs, user can press F1 and can know the cause of the problem in emulator window.

10 Project Settings

To modify parameters, select *Project →Settings…*, and the window shown in Figure 34 is displayed with several tabs.

Click on the *Restore Default* button if you wish to restore settings to the default:

10.1 Layout Setting

Designer *layout:* Enables snap lines or a grid on the designer window.

Gridsettings*:* If Grid layout selected, Grid settings sets the width and height of grid in designer window.

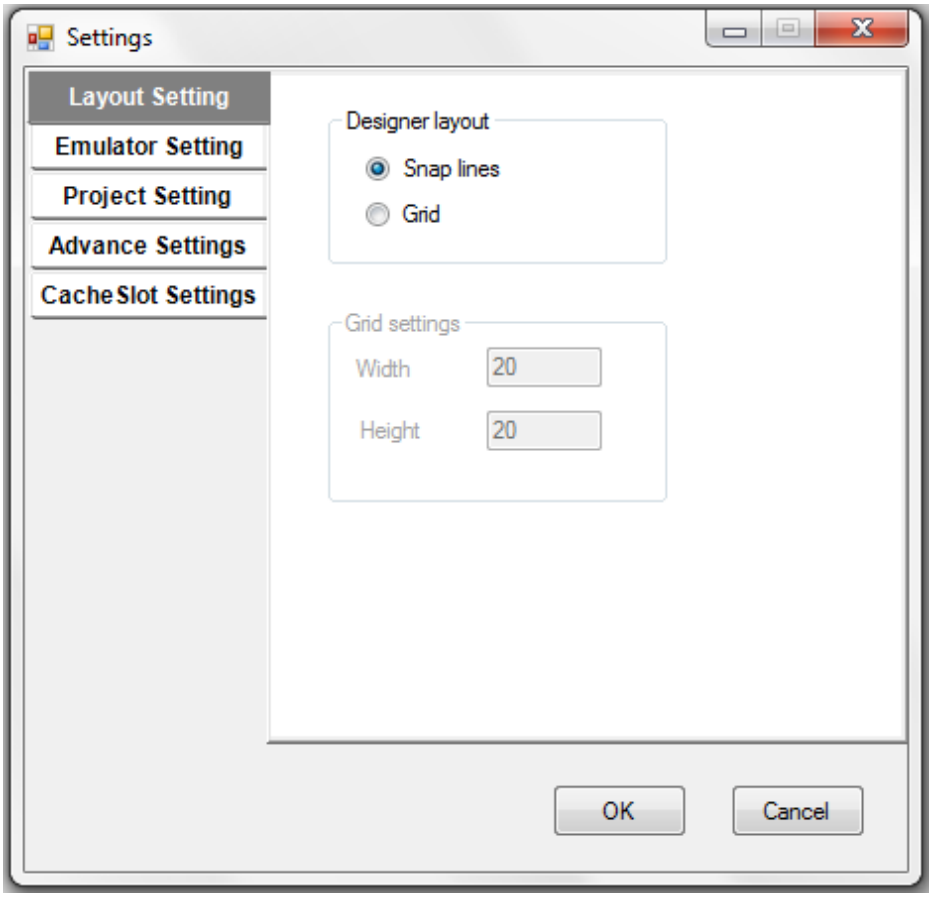


Figure 34: Project Layout Settings Tab

10.2 Emulator Setting

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 windowFigure 35.

Binary Data Identifier:

Allows to set 4 bytes of binary data identifier for flash binary.

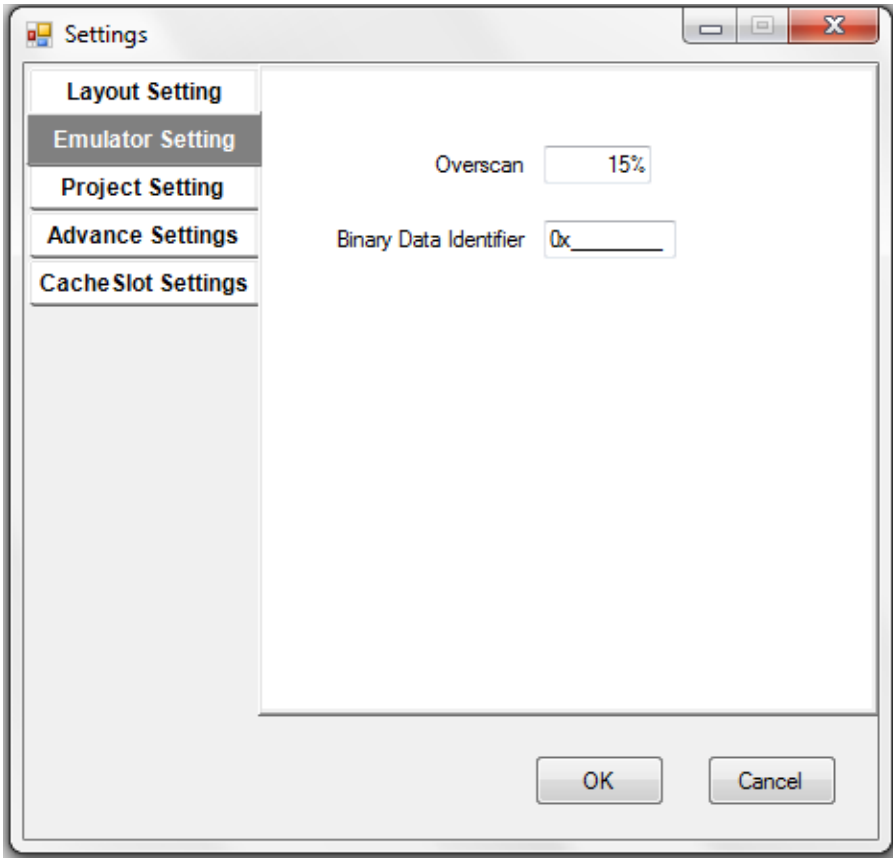


Figure 35: Project Emulator Settings Tab

10.3 Project Setting

Compile settings:

Used to select a font bit resolution that will affect the quality of the fonts and the amount of memory needed for storage:
1 bit fonts – fonts will be packed in binary data with 1-bit resolution

4 bit fonts – fonts will be packed in binary data with 4-bit resolution (means alpha channel information is being used)
8 bit fonts – fonts will be packed in binary data with 8-bit resolution (means alpha channel information is being used)

This is the number of bits used when bitmapping the system fonts, allowing for storing and DDR2 bandwidth savings.

Default Page Size*:*

Sets the default width and height that the created pages will have. Later, this can be changed individually for each page by modifying the *Size* property within the *Property Navigator* panel.

OSD Resolution Size:

Sets the OSD resolution size of the overlaid OSD on the active video. Note that the OSD Resolution size and page size are not related in any way. It is possible to have multiple different sized pages in the same project and move them to the required position within the OSD.

Flash Size:

An option to add padding to compressed flash file to be 8MB and 16MB shall be available. Not selecting any of these options will leave the flash at the same size when compressed with no padding.

Splash Page:

The Splash page data shall have the option to be compressed or uncompressed. Data compression option can be set in project settings.

Splash Resolution:

Resolution settings for splash page (Width & height) can be done in project settings.

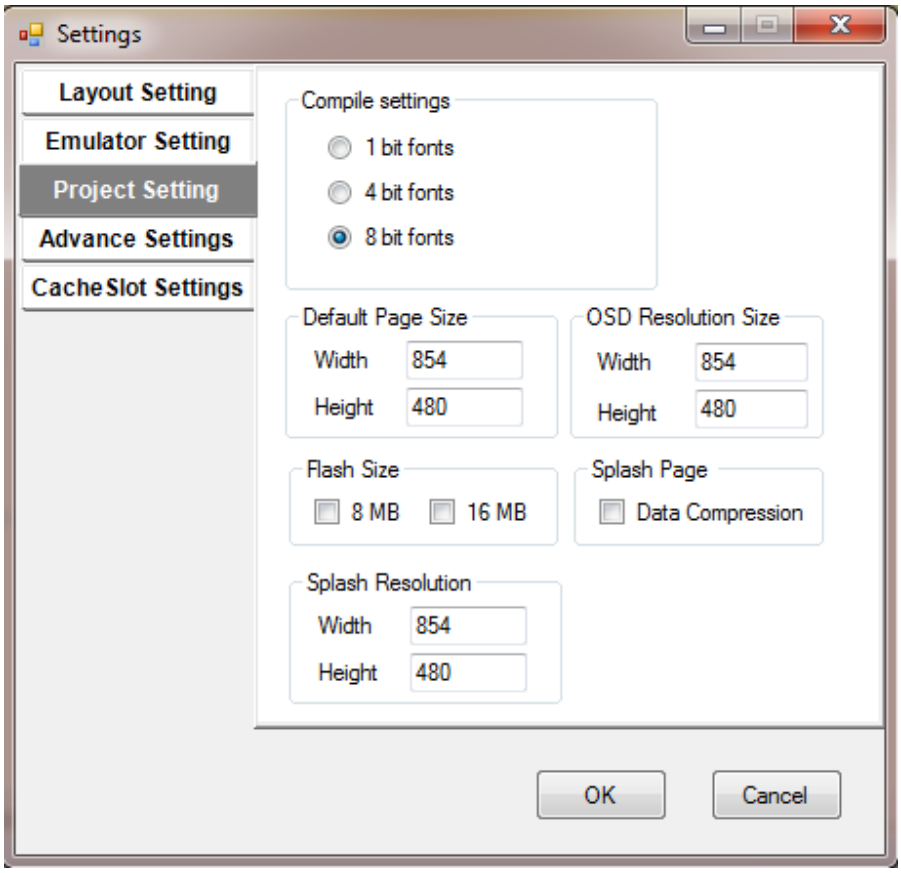


Figure 36: Project Settings Tab

10.4 Advanced Settings

MaximumBandwidth–sets the maximum bandwidth in bytes per second allocated to the OSD. The default value is 600000. Note that DDR2 memory is shared with VSP, ensuring VSP gets a minimum priority bandwidth.

DDR2Size–specifies the size in bytes of DDR2 memory. Page buffering is used if the OSD memory requirement exceeds this value.

MinimumPageBuffer–sets the minimum number of pages resident in DDR2 memory when the page buffer is active.

MaximumCacheSlot– sets the maximum cache slot number for OSD project. It depends upon the number of components and their regions present in the project. Default value is 1024. If the OSD project needs more than the MAX_CACHE_SLOT given, the error will reports with MAX_CACHE_SLOT number needed.

CacheStartAddress - OSD_CACHE_ADDR can be a configurable hex value by user in project setting(in Advanced tab).
default value - 0x0610800

Virtual Heap Size -Virtual heap size can be a configurable value in bytes by user in project setting(in Advanced tab). An option(ComboBox) should be available for virtual heap size which allow user can either enter the custom value or choose an 'auto' option.

default value - 100720 bytes

Note: when Auto options choosen, Blimp will calculate the virtual heap size of the project automatically.

Transparent Color - Transparent Color can be a configurable value in hex by user in project setting(in Advanced tab).
default value - 0x00a050a0

Time Accuracy - Time Accuracy can be a configurable value in ms by user in project setting(in Advanced tab).
default value - 500ms.

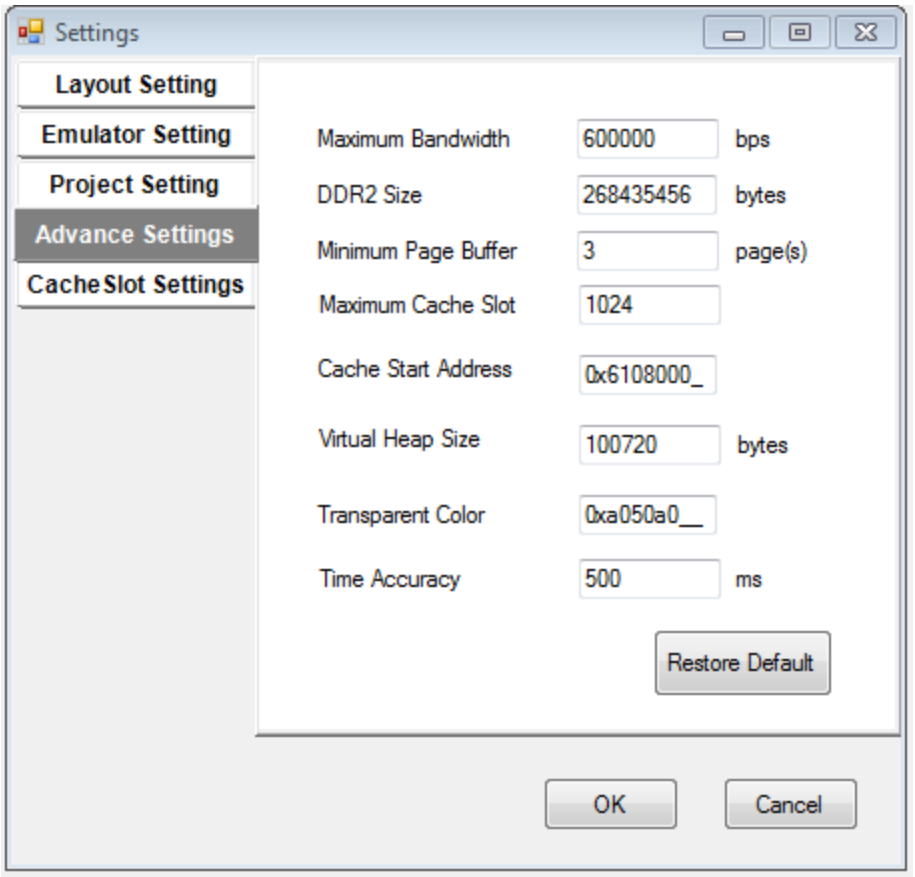


Figure 37: Project Advanced Settings Tab

Click on the *Restore Default* button if you wish to restore settings to the default:

Maximum bandwidth: 600000

DDR2Size: 256MB

Minimum page buffer: 3

Maximum Cache Slot: 1024

Cache Start Address: 0x0610800

Virtual heap size: 100720 bytes

Transparent color: 0x00a050a0

Time Accuracy: 500ms

10.5 Cache slot Settings

CacheSlot settings allow user toconfigure the cache slot number of OSD components based on their OSD design to avoid cache miss. An option(ComboBox) should be available for cacheslot which allow user can either enter the custom value or choose an 'auto' option for each component.

Note: The info tip will shown when Auto options choosen "Blimp will count the maximum no. of that component per pages".

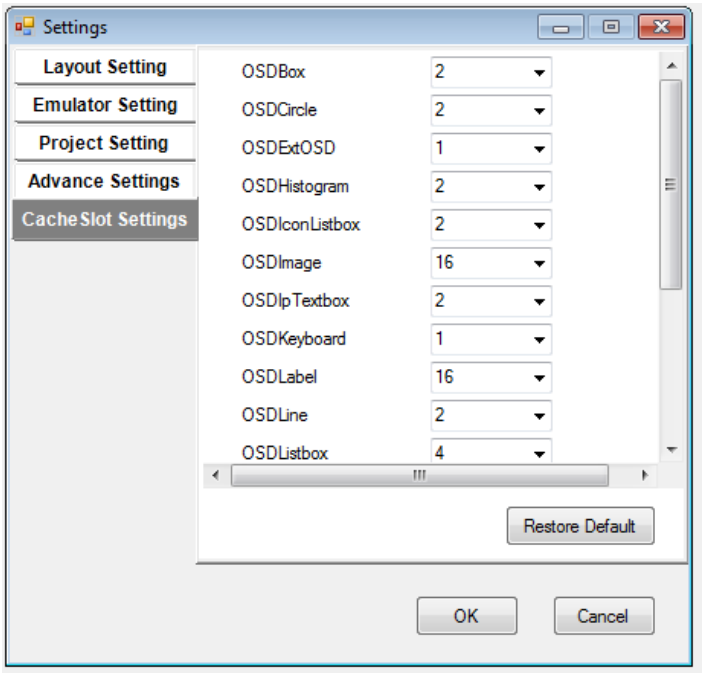
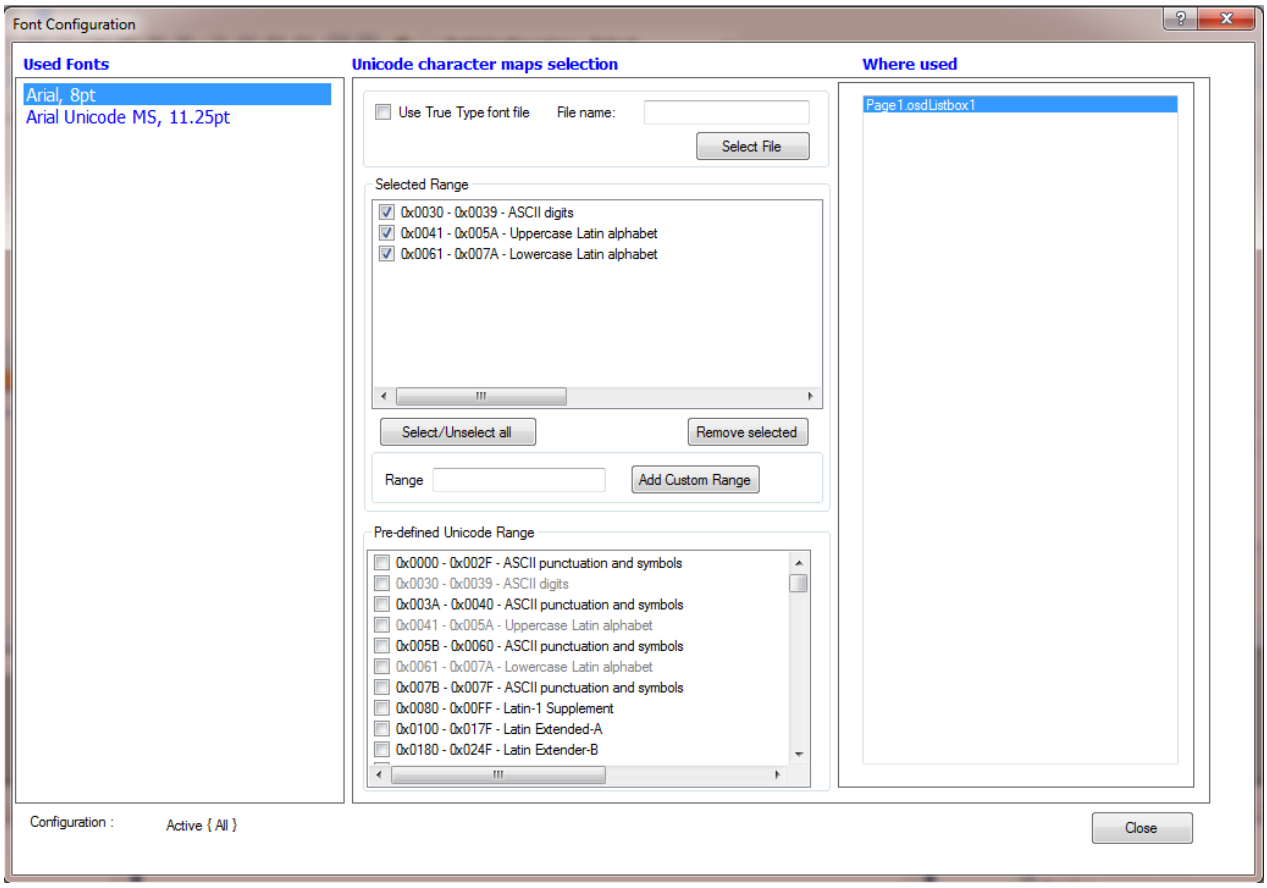


Figure 38: Project Cache Slot Settings Tab

11 True type font support on MCU side

The support of font files of true type will allow OSD to draw any size of a specific font on the fly in run-time using freetype library.

When click the Use True Type font file checkbox font file linked to the selected font Arial



As shown in the GUI window pointer, there is a selection to choose True Type fonts for each font used. It will allow for a file to be opened and file name will be shown in textbox.

When ‘User selcts True Type font file’,the default font file will be loaded automatically from Windows with default name filled in file name textbox as per file name in Windows. When it is selected,Blimp will search for all other fonts using same True Type font and replace all other font of same name to also use font file

After font selection, this font file will be copied and store in the project folder. When unselected, other fonts using this font file will not change, but if no other fonts are using this file, then it will be removed from the project folder.

While building the project, the font file will be stored in flash with compression applied by default. The font can be loaded to drr2 memory entirely to allow drawing of strings.

Currently the font file type supported shall be TTF. The type of font files supported as per the FreeType library version 2.5.3 In future other file type will be added as per the request.

By using a font file for a sepcific font, the Blimp code window will be used to specify a different size at run-time.

For sample, a test string as mentioned below can be used with many sizes

The quick brown fox jumps over the lazy dog. 0123456789
The quick brown fox jumps over the lazy dog. 0123456789
The quick brown fox jumps over the lazy dog. 0123456789
The quick brown fox jumps over the lazy dog. 0123456789
The quick brown fox jumps over the lazy dog. 0123456789

Code window usage example:

```
// Based on the index the titleLabel size will be set.
 public void OnInitLoad()
    {
    titleLabel.Text="The quick brown fox jumps over the lazy dog. 0123456789";
    }
    public void Dispose()
    {

    }
    private void Page1_RemoteKeyPress(Byte* keyCode, Boolean* cancel)
    {

        switch(*keyCode)
        {
        case 49:
             titleLabel.FontSize = 14.0F;
             break;
        case 50:
             titleLabel.FontSize = 11.5F;
             break;
        case 51:
             titleLabel.FontSize = 12.5F;
             break;
        default:
             titleLabel.FontSize = 10.5F;
             break;
         }
    }
```

12 Multilanguage String Configuration

See Blimp manual for details on how to define language strings.Character Map selection

13 Build Configuration

Build Configuration module is used to define and store the pre-processor name and value. User can add / edit the multiple build configurations. Each configuration has its own and common preprocessor definitions. If more than one configurations added then Active build configuration can be changed using Build Configuration toolbar selection from the main window. The pre-processor names can be usedin code window using #if and #ifdef

13.1 Configure the Build configuration

13.1.1 To Open the Build Configuration window

Click the Build menu and select the Configuration Manager… menu. Build Configuration window is shown as below.

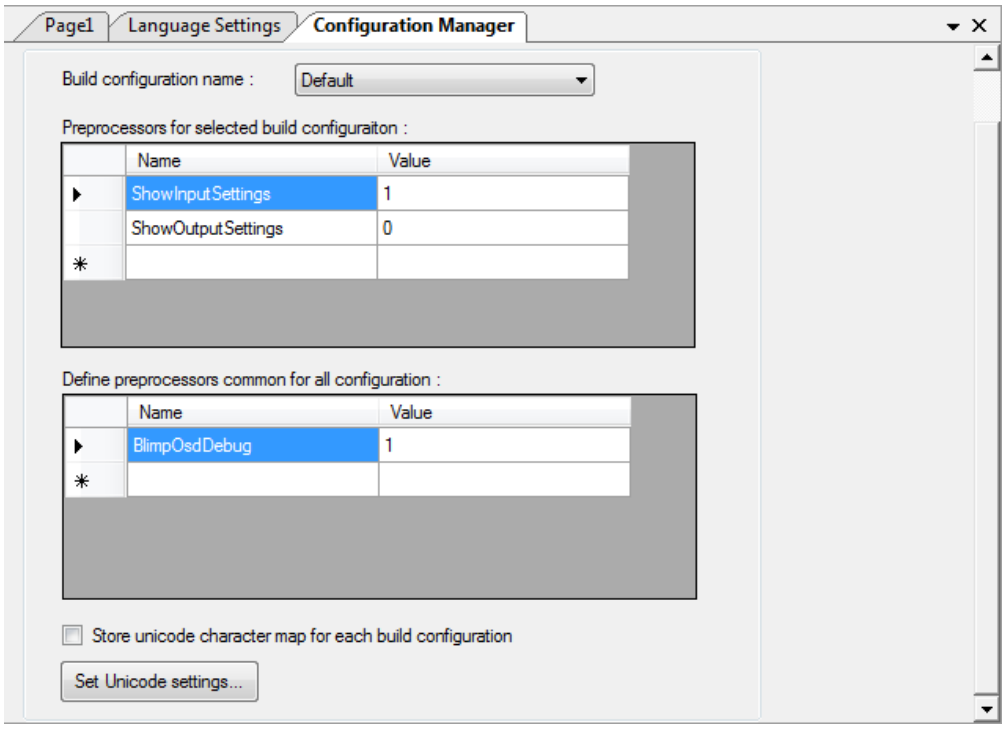


Figure 39: Build configuration window

13.1.2 Add or edit a Build configuration

- User shall be able to add a new build configuration, modify its name or delete a build configuration.
- A limit of 10 configuration can be defined.
- A minimum of 1 configuration must be defined.

For add a Build Configuration, select Build configuration name list and select <New..>Option.

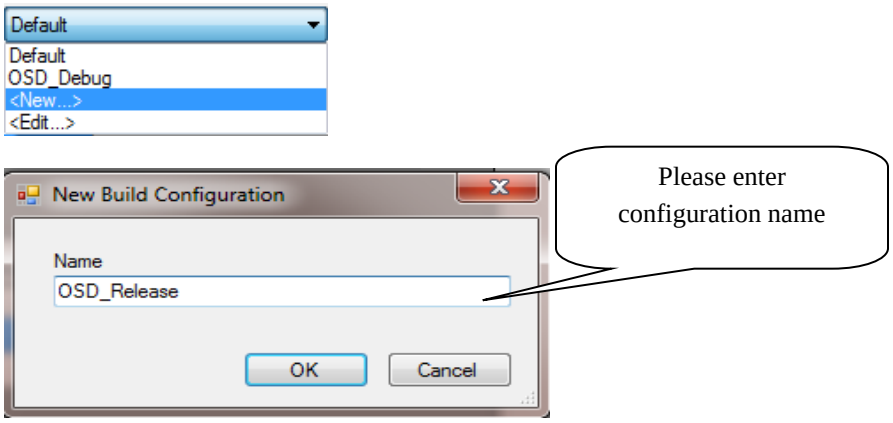


Figure 40: Add Build configuration window

For edit or remove a Build Configuration, select Build configuration name list and select <Edit..>Option.

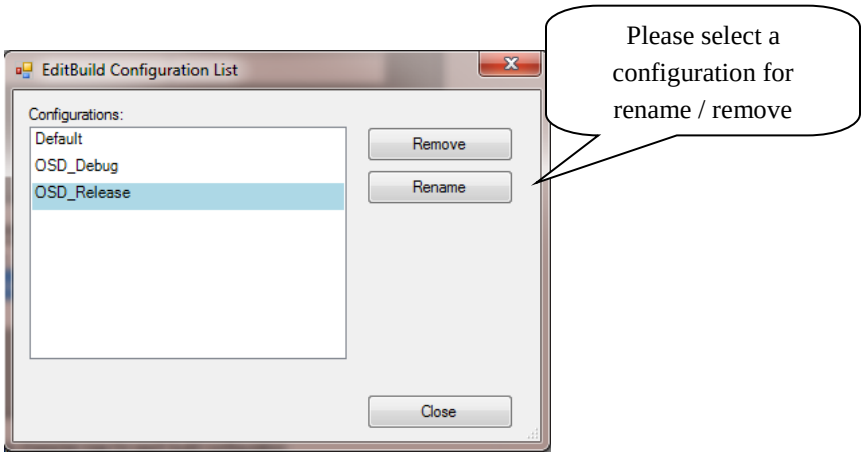


Figure 41: Edit or RemoveBuild configuration window

13.1.3 Configure the pre-processor

Add preprocessors to first list which are unique for each build configuration. Add preprocessors to second list which are common for all build configurations. List will accept preprocessor name and value. Value can beeither in numerical or string.

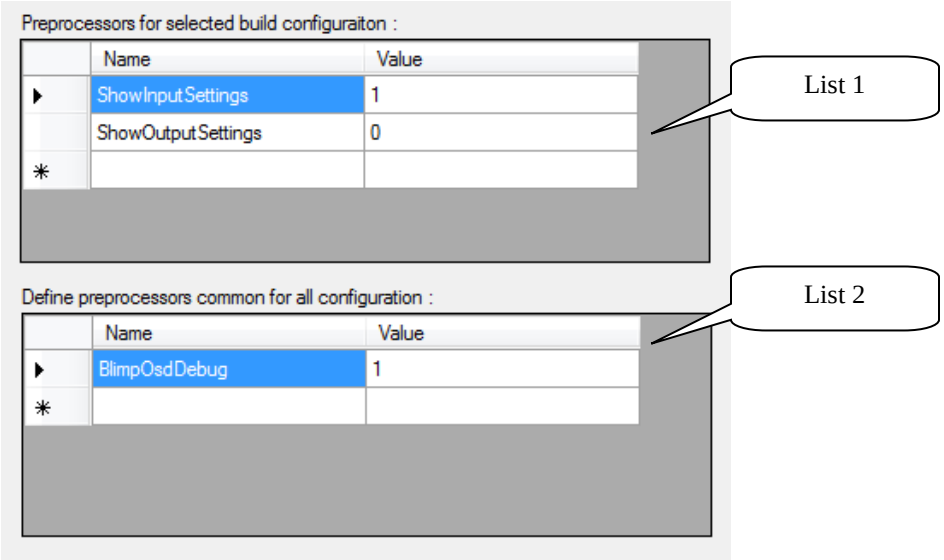


Figure 42: Configure pre-processor window

13.1.4 Active build configuration

User shall be able to select the active configuration from the Blimp tool bar through a drop-down list.

Note that when building or Emulating OSD project the preprocessor names and settings of active build configuration can be used.

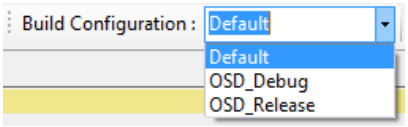


Figure 43:Active build configuration window

13.1.5 Update Unicode character map for Build configuration

- A check box in the build configuration window shall allow user to specify if Unicode character map configuration will be defined for each build configuration.

- If selected, Blimp shall store a Unicode character map for each configuration.
- If unchecked, a warning message shall be displayed saying "Warning! removing this option will delete configuration specific data for Unicode character map configuration. Are you sure?" If no is selected, cancel action. If yes selected, only one Unicode character map is kept and others are deleted.

User can configure the Unicode character map for each build configuration as shown in Figure 44

Open the Unicode settings under the project menu

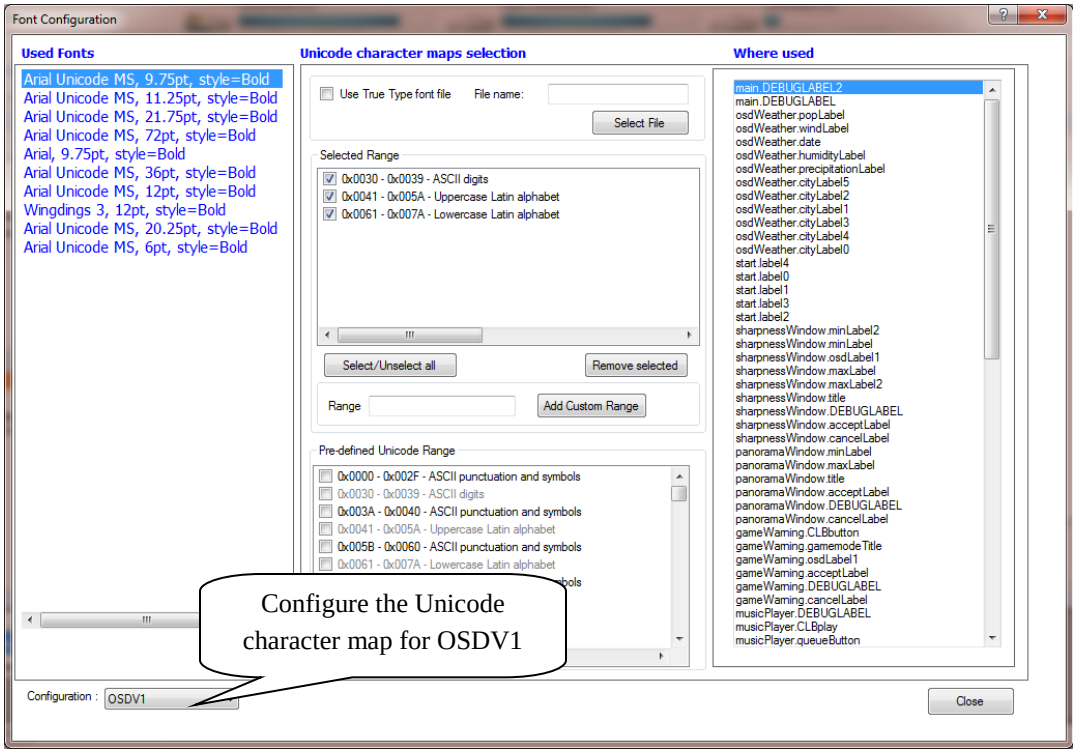


Figure 44:Unicode character map for build configuration window

If the check box – "Store Unicode Character map for each build configuration" which is available in Configuration Manager is checked,the configuration dropdown list in the Unicode Settings will be enabled to store the Unicode for each configuration. Otherwise the dropdown will be disabled.

Code window usage example:

```
// the ShowOutputSettings is the preprocessor which is defined in the
build configuration
```

```
#if ShowOutputSettings
    /*Using this preprocessor user can compile out codes/API related
output settings in all Osd pages.*/

#endif
```

14 Font Style Configuration

Multiple font styles can be assigned along with the different languages in a single component or multiple components during run time

14.1 Configure Font Style

14.1.1 Open the font style configuration window

Click the project menu and select the Font Style Configuration menu.

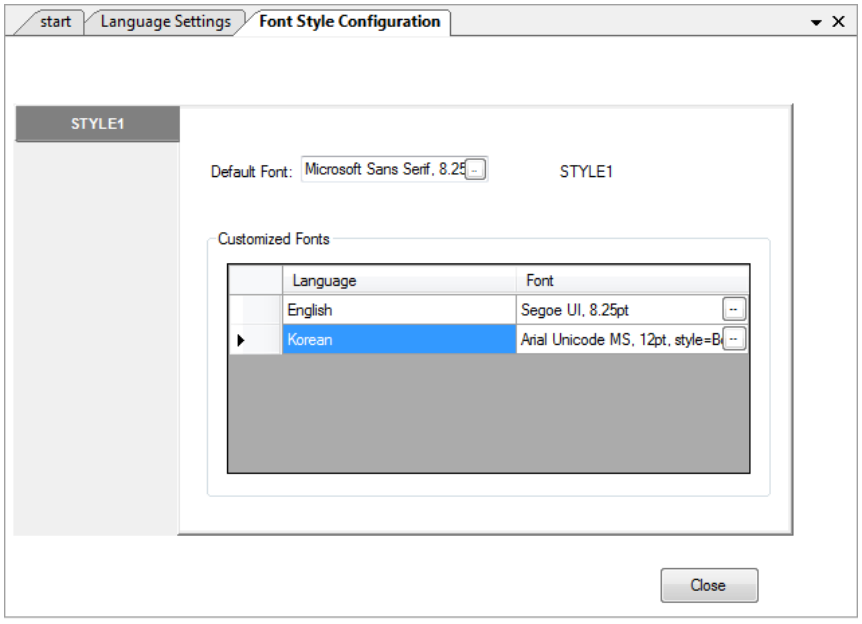
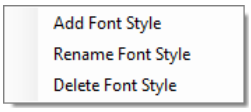


Figure 45:Font style configuration window

14.1.2 Add or edit or delete font style configuration

Select font style name and right click on the tab to add/rename/delete font style configuration.



For adding font style, click “Add Font Style” option in the list.

The font style name window is shown as below.

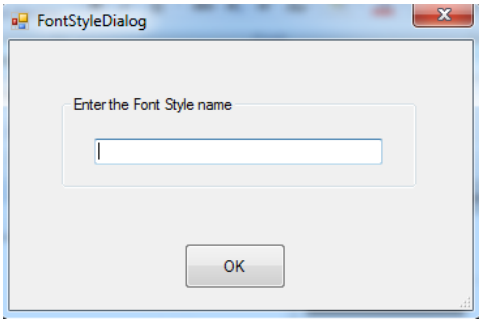


Figure 46:Add font style configuration window

Font style name should be unique and not null (empty)

For rename font style, click “Rename Font Style” option in the list.

The rename font style window is shown as below.

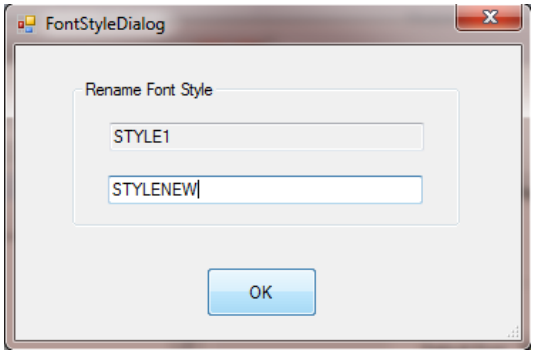
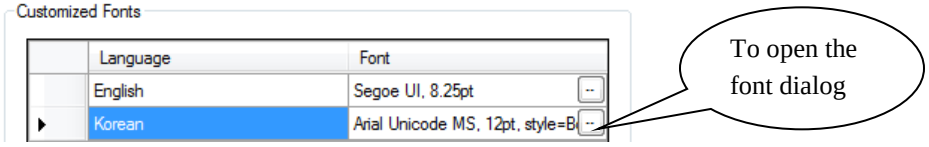


Figure 47:Edit font style configuration window

To delete a font style, click “Delete Font Style” option in the list, The corresponding font style will be deleted.

14.1.3 Add or modify a font in different languages

Click the corresponding language button under the font column, the font dialog window will be opened . User can select a font name, font style and font size from font dialog.



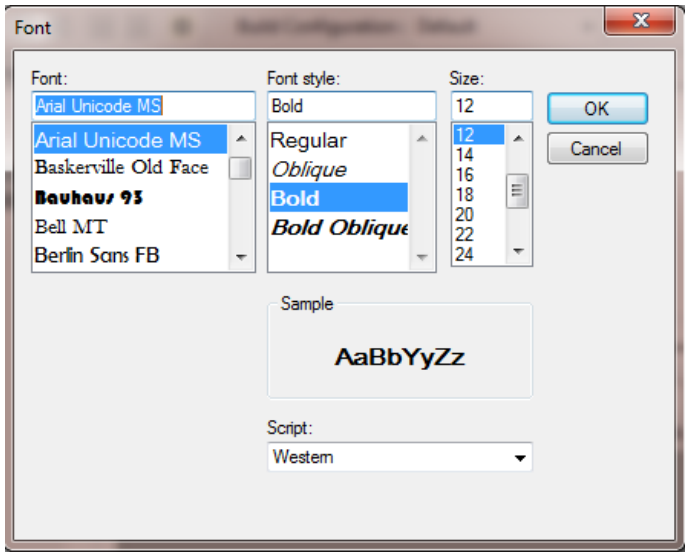


Figure 48:Update fonts in font style configuration window

When the font is not specified in customized font window, the default font will be updated in the corresponding language.

14.2 Configure font style in OSD component

User can set the font style in Font property for one or more components.

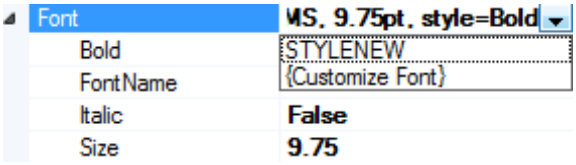


Figure 49:Sets the font style in component property window

User may select a font using “Customize Font” option or font style configuration

15 Image Library

Image library is the database which is used to add images for OSD components (like *OSDImage*, *OSDIconlistbox* and *OSDMultiColumnListbox*). User can maintain images which are going to be used for creating OSD project.

15.1 Add or remove the images in Image library

To add or remove the images in Image library, Open the image library option in project menu.

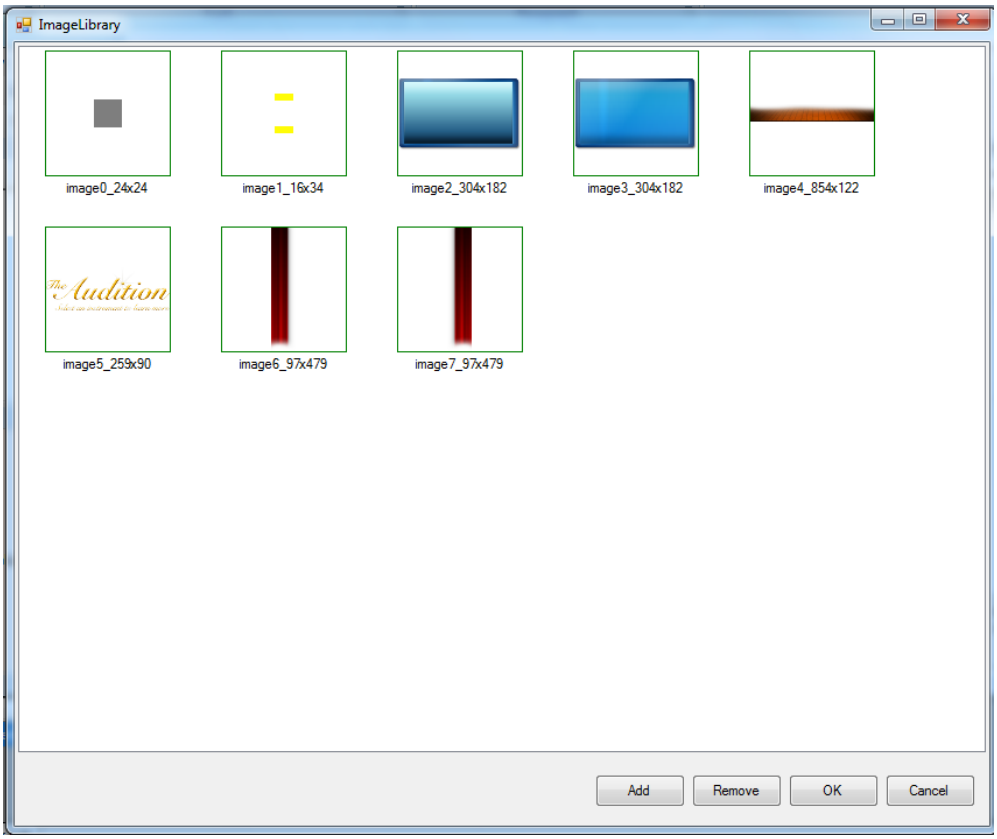


Figure 50: Image library window

15.2 Access the Image Library in OSD Components

For OSDImage component, we can add an image from image library in “ImageSettings” property.

Image settings property windows is shown as below.

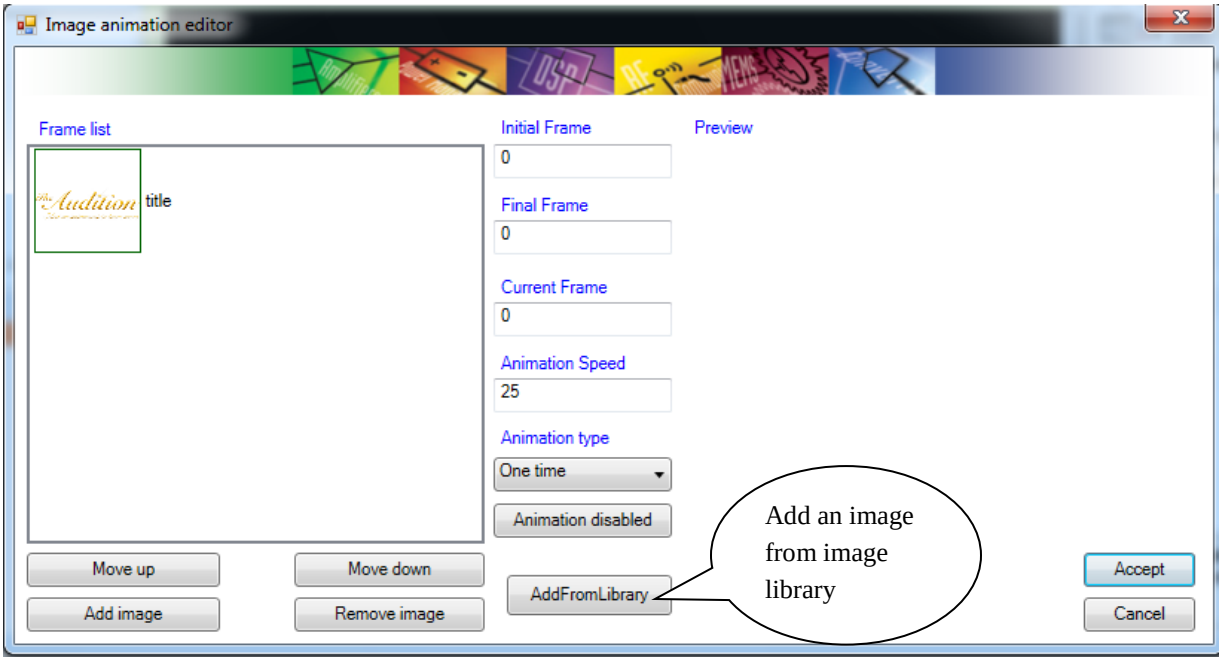


Figure 51: Image animation editor window

For OSDIconListbox or OSDMultiColumnListbox component, we can add an image from image library in "IconSettings" property.

An icon settings property window is shown as below.

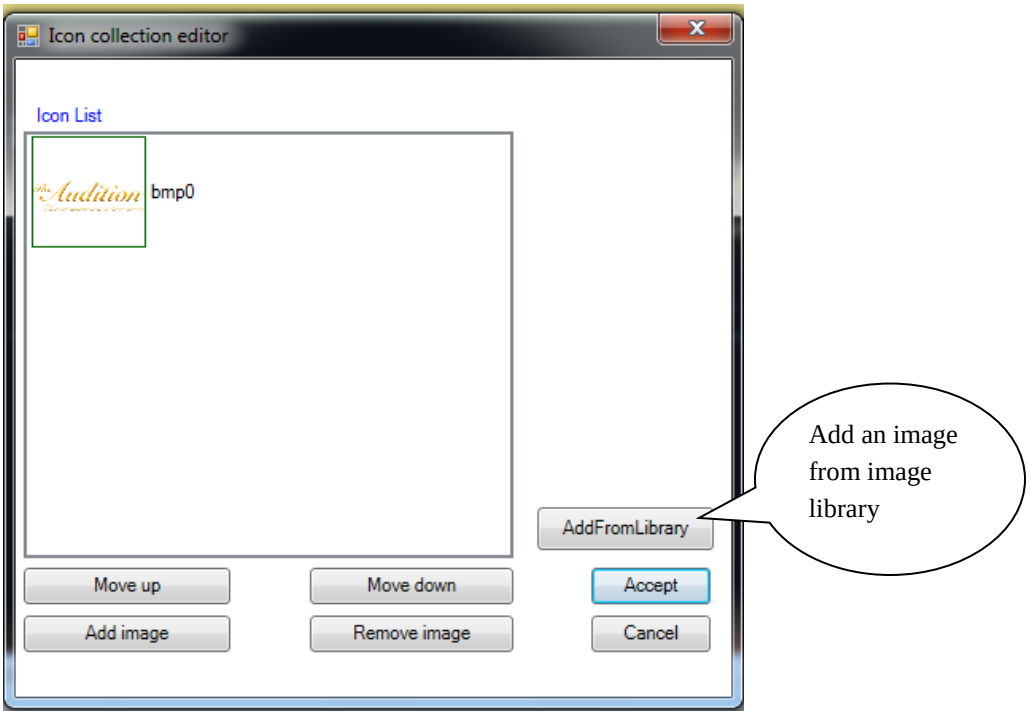


Figure 52: Icon collection editor window

16 OSD Integration

See Blimp user manual section 9 for all steps to integrate Blimp output into MCU.

This section contains specifics to ADV8002 for DDR2 memory

16.1.1 SPI Reading/Writing Functions

The ADV8002 uses SPI read and writes to and from DDR2 memory. The functions defined in Blimp user manual section 9.2.1 must exist to be able to build.

16.1.2 Initialize DDR2 Controller

Executes HAL I2C write to initialize DDR2 memory access.

From the provided*osd_control.c*, follow these steps to initialize the ADI evaluation board DDR2.

- Set SPI slave mode, 0x0c -> CPOL/CPHA = 1/1.
- Invert input DE polarity.
- Initialize DDR controller, 400 us for 222 MHz clock.
- Set the SPI bus to auto.

17 Tutorial for Basic OSD Example

In this section, a one page basic OSD is designed from scratch to show the different components of the OSD, and how the interactions between them and the user can be built through the code window of the Blimp.

Follow these steps to create the basic OSD.

- Open Blimp and create a new project. Select ADV800x OSD Application and click on the Create button.
- Click on the Yes button on the “Do you want to create a default empty page” pop-up window.
- On the New item window, click on the Add button to create a new page with the default name of page1. Now you can start adding components onto the blank canvas.
- Drag and drop one OSDLabel (named OSDLabel1 by default) and two OSDListbox (named OSDListbox1 and OSDListbox2 by default) components into the design, and place them as shown in Figure 53.

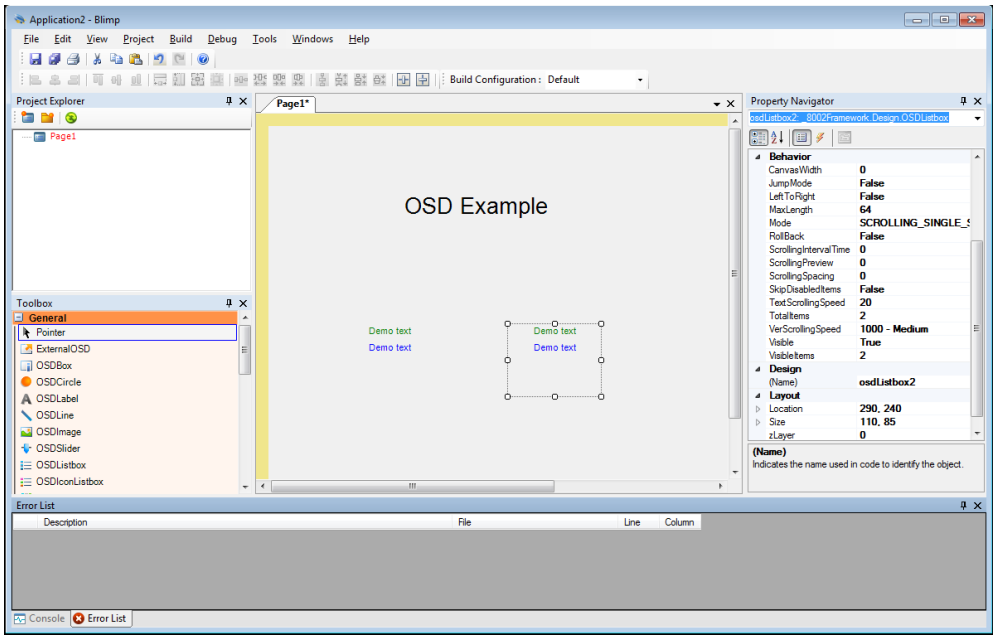


Figure 53: One OSDLabel and Two OSDListbox Components Inserted on Blank Canvas

Notes:

- You can rename these components on the Property Navigator to make them more descriptive.
- As you move the component around the canvas, smart guides appear to assist you with placing the component.
- Using the Property Navigator panel, change the properties of the inserted components to match those inTable 50.

Table 50: Components Inserted on OSD Design and Associated Properties

Property	OSDLabel1	OSDListbox1	OSDListbox2
Font	Arial	Default	**Default**
Font Size	20	15	**15**
Text	OSD Example	-	**-**
FgColor	**White**	-	**-**
Default Text Color	-	Default	**Orange**
Highlighted TextColor	-	Default	**Cyan**
Selected TextColor	-	Default	**White**
TotalItems	-	4	**6**
Visible Items	**-**	**4**	**3**

The screen should look like the sample in Figure 54.

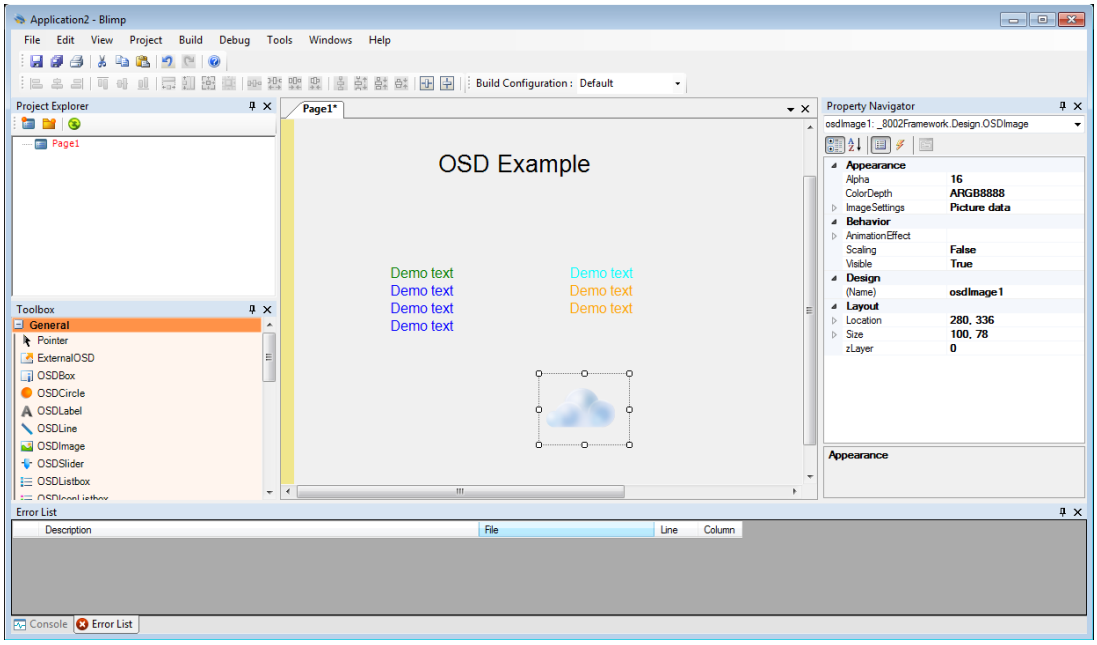


Figure 54: Components Properties Changed

- To finish the OSD layout, follow this step to insert one OSDImage component, which will behave as an animation.
- Drag and drop this component into the canvas.
- Use the Image property to link an image to the component.

- Add the images to the component and change ImageSettings property as shown in Figure 55 to make an animation.

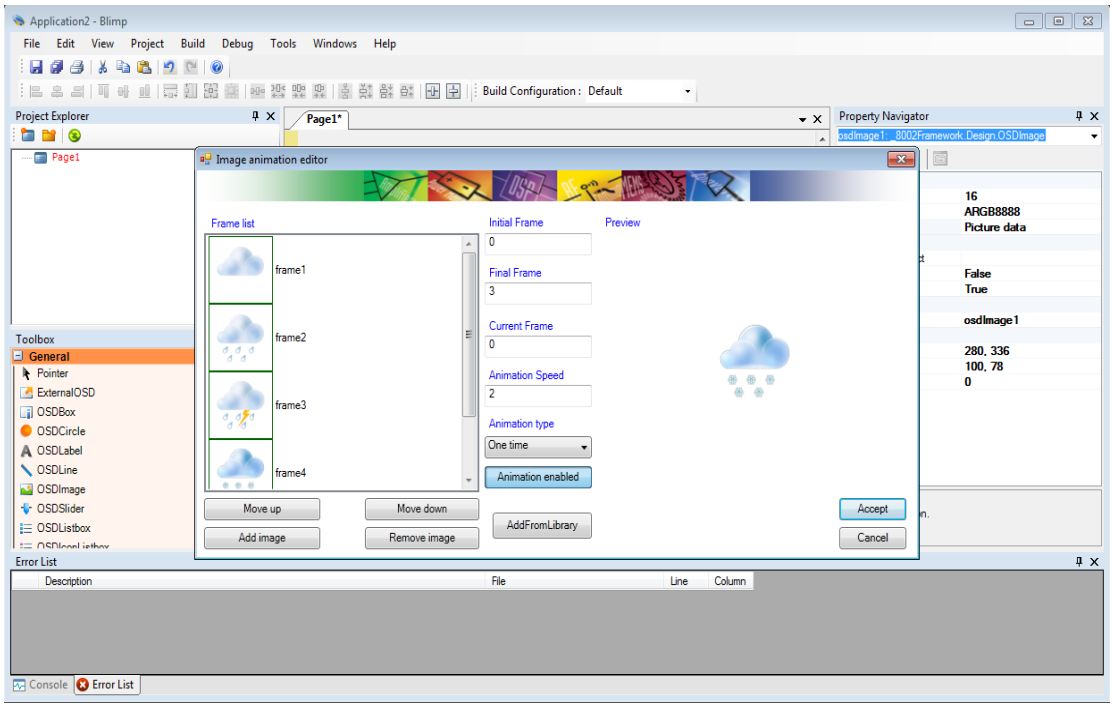


Figure 55: Inserting Animation into Design

You have created an animation with five frames. The OSD layout is now complete.

- Follow this step to visualize the layout using the emulator window.

Select *Tools -> Emulator* (or press*F5*). This compiles and links the project, and it is visualized on the emulator window. The OSD design should look likeFigure 56.

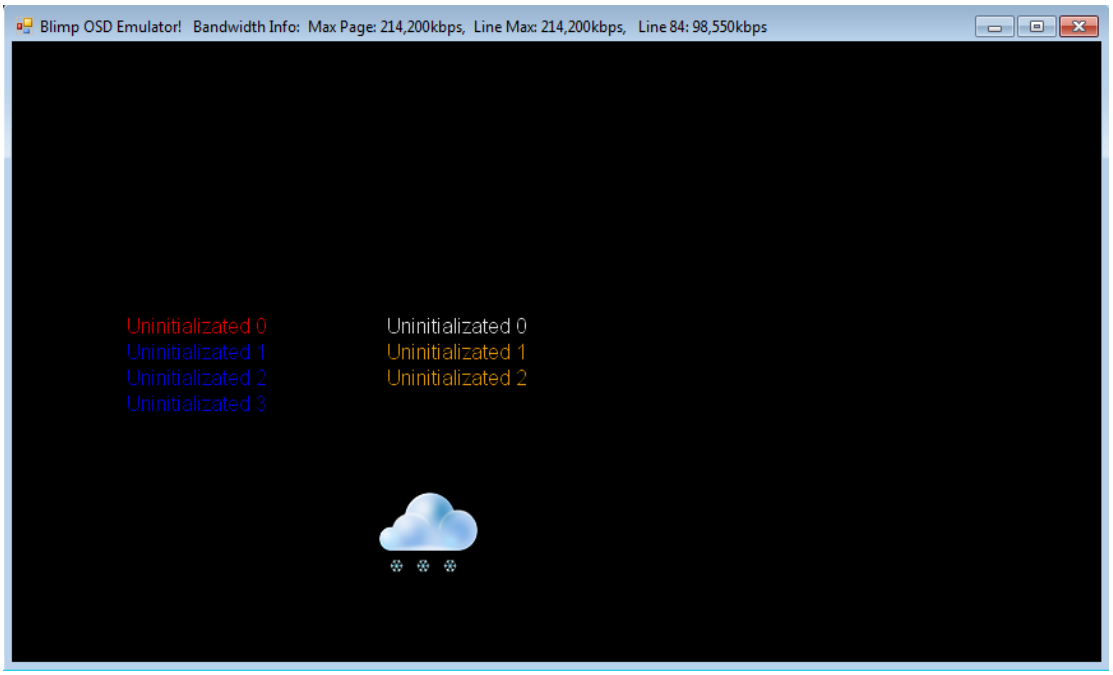


Figure 56: OSD Design View on OSD Emulator Window

- Now you can start defining the way you want the OSD to respond to user interactions by using the code window.

  Right click on Page1 of the Project Explorer panel and select View code.

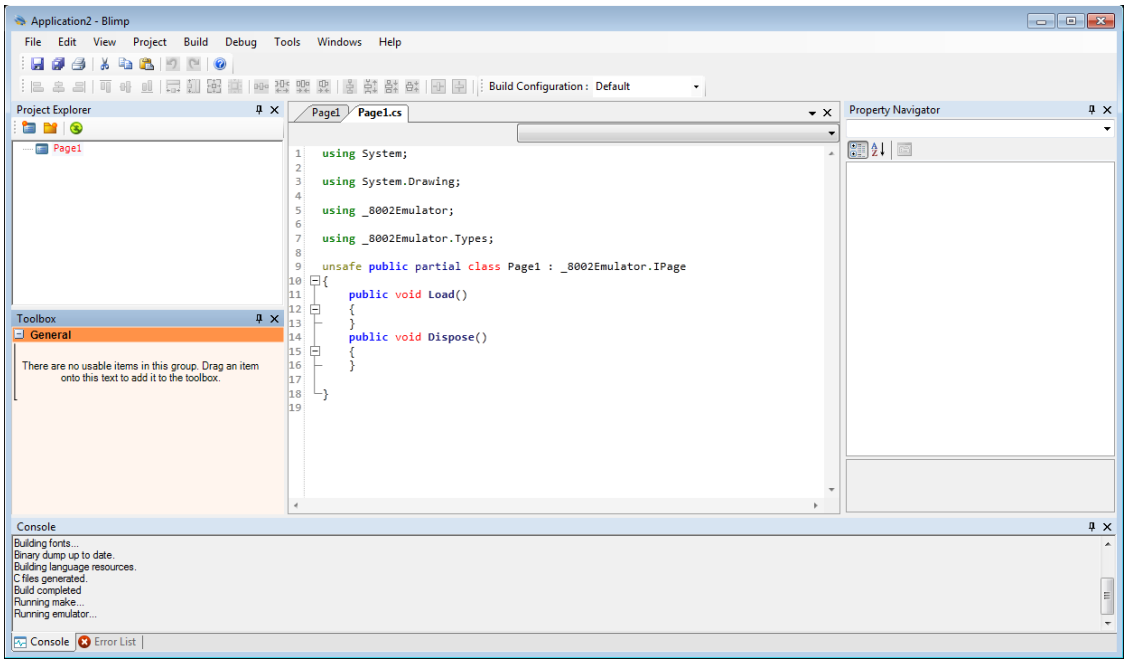


Figure 57: Code Window of Example1.osl

- First of all, some component properties need to be set in order to define the initial state of the OSD. This can be done through the Load() method.

```
public void Load()
    {

    //Make osdlistbox2 and image not visible on startup
    osdListbox2.Visible = false;
    osdImage1.Visible = false;

    //Note how the properties can be defined either through the GUI or
code
    //Initialize items on the list. Otherwise, they will be set to
"uninitialized"
    osdListbox1.ItemText[0] = "Input Select";
    osdListbox1.ItemText[1] = "Video Settings";
    osdListbox1.ItemText[2] = "Settings";
    osdListbox1.ItemText[3] = "Information";

    //Set the focus to the osdListbox1 component. Since it is a mono-
page design, no focus on the page
    OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
    }
```

- Create the method which will handle the event triggered from the keyboard input.
- Select the osdListbox1 component.
- Press the Events button in the Property Navigator panel.
- Double click on the RemoteKeyPress text. Automatically, the code window opens and a method called osdListbox1_RemoteKeyPress is inserted.

  This name can be modified if desired, but will need to match the name defined in the Property Navigator.

  Figure 58 shows how the events property of *osdListbox1* should now look.

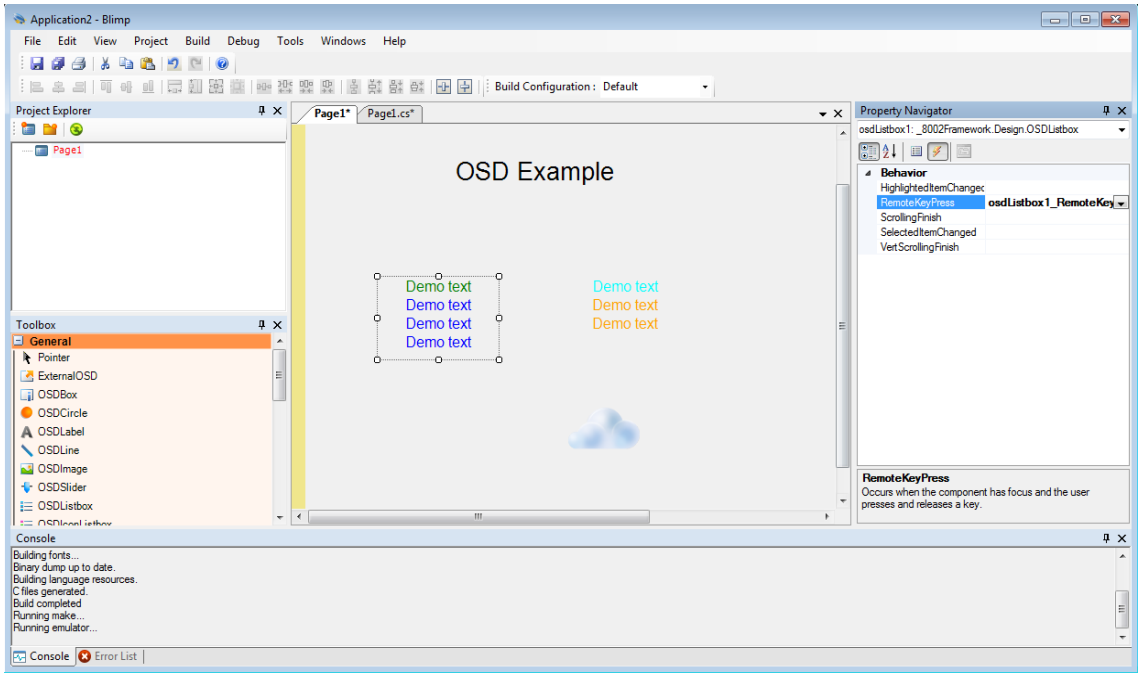


Figure 58: RemoteKeyPress Event and Associated Method

The new method added into the code is used in this example to handle the captured keyboard input so you can move from the main menu (*osdListbox1*) to the submenu (*osdListbox2*), when pressing the right arrow key. Note that the*keycodes* are hardcoded in Blimp and you cannot assign them at this moment.

```
//    The LEFT ARROW key = 37,
//    The UP ARROW key  = 38,
//    The RIGHT ARROW key  = 39,
//    The DOWN ARROW key = 40,
    private void osdListbox1_RemoteKeyPress(Byte *KeyCode, Boolean
*cancel)
    {
        if (*keyCode == 39) //Right arrow key pressed
        {
            //Bring focus to the list on the right
            OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox2);

        }
    }
```

- Follow the same process for osdListbox2.

  The following code is used for this method.

```
private void osdListbox2_RemoteKeyPress(Byte *keyCode, Boolean *cancel)
    {
       //Disable these here because we will be enabling them on the next
steps...

       osdLabel1.Visible = false;
       osdImage1.Visible = false;

        if (*keyCode == 37) //Left arrow key pressed
        {

            //Bring focus to the list on the left and hide osdListbox2
            OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox1);
            osdListbox2.Visible = false;
        }


    }
```

  At this stage, you should be able to move between both lists by using the arrow keys.

- The next step is to initialize the elements in the list on the right. This initialization will be one or other text string depending on the item on the osdListbox1 when you moved right to the second list.

  To do this, you create and use a custom defined method as follows.
- Modify the previously presented code to:

```
private void osdListbox1_RemoteKeyPress(Byte *keyCode, Boolean *cancel)
    {

        if (*keyCode == 39) //Right arrow key pressed
        {
            //Bring focus to the list on the right
            OsdApi.ADIAPI_OSDEgSetFocusComponent(osdListbox2);
               //Call to user-defined method ! Note how we can pass
values, the index in this case
            configListboxItems(osdListbox1.SelectedIndex);
```

```
        }
    }
```

- Add the custom method as below. Depending on the expected application, the behavior of the list can be changed on execution time. That is, since several video filters could be applied at the same time to the video, the case 1 implements a multiple selection list, instead of the single selection defined at case 0 (default option).

```
    private void configListboxItems(Byte index)
    {
        switch(index)
        {
            case 0:

                //Note how the user can scroll down on this list of
items
                osdListbox2.TotalItems = 6;
                osdListbox2.ItemText[0] = "HDMI 1";
                osdListbox2.ItemText[1] = "HDMI 2";
                osdListbox2.ItemText[2] = "HDMI 3";
                osdListbox2.ItemText[3] = "CVBS";
                osdListbox2.ItemText[4] = "S-Video";
                osdListbox2.ItemText[5] = "Component";
                osdListbox2.Visible = true;
                break;

            case 1:

                osdListbox2.TotalItems = 3;
                osdListbox2.ItemText[0] = "Noise Filter";
                osdListbox2.ItemText[1] = "Sharpness";
                osdListbox2.ItemText[2] = "Color Enhancement";
                osdListbox2.Visible = true;
                break;
        }
    }
```

- You should now be able to move from one list to the other.

  Note thatthe Input Select submenu items scroll as you move down the list.

You can have several items selected (use the spacebar to select any item on a list) at the same time on the Video Settings submenu but not on the Input Select submenu.

- Now you can add one more event and custom method, which will account for the selection action to any list item.

```
private void osdListbox2_SelectedItemChanged(Byte index,Boolean
newStatus)
    {
        //Make visible and wnable image when the user presses the
spacebar
        osdImage1.Visible = true;
        osdImage1.Enabled = true;

        //Call a custom method to turn on the textlabel
        setText(newStatus);
    }

    private void setText(Boolean flag)
    {

       if(flag)
            osdLabel1.Text = "Option Enabled";
        else
            osdLabel1.Text = "Option Disabled";
    }
```

Now you should be able to see the animated star moving around when you press the space bar on any item of the submenus. This is shown on Figure 59.

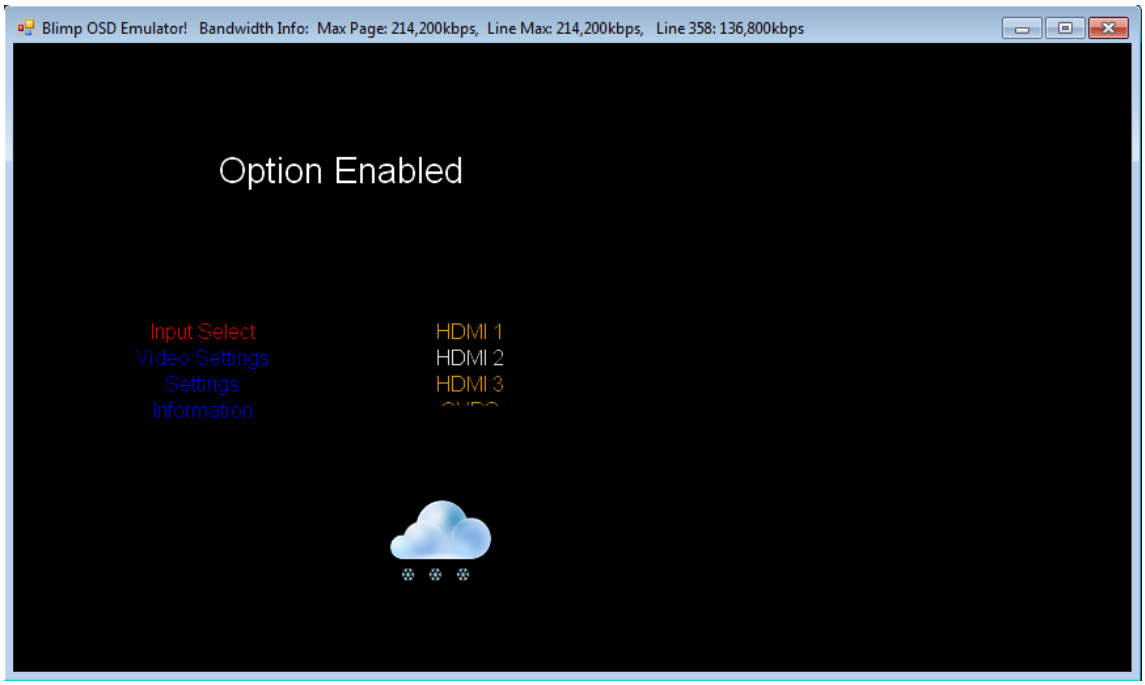


Figure 59: Final View of Example1

18 Features in Emulator Window

'Right click' in emulator window will provide the list of emulator features as shown inFigure 60.

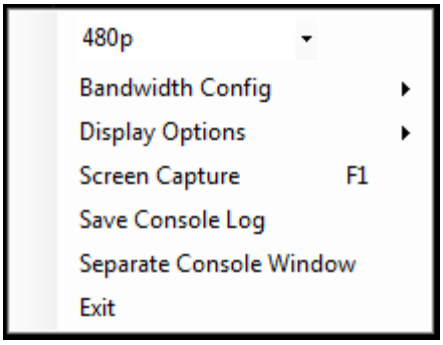


Figure 60: Emulator features

18.1 Screen capture

When you press F1, a screenshot of the current emulation window is taken and stored in the *Release/ScreenCapture* folder. Screen capture feature also available in the emulator window right click menu.

18.2 Save Console log

User could save the emulator console log information in to the text file by pressing the ‘Save Console Log” menu.

18.3 Separate Console Window

When you press *F2*, the console log information will be displayed in bottom of emulator window. ‘Separate Console Log’ option will separate out the console log window from emulator winow.

18.4 DDR2 Bandwidth

Blimp emulator window will display the DDR2 bandwidth information in kbps.

line Max – Max line bandwidth on that page

Line – By moving the mouse, we can get the bandwidth for each line.

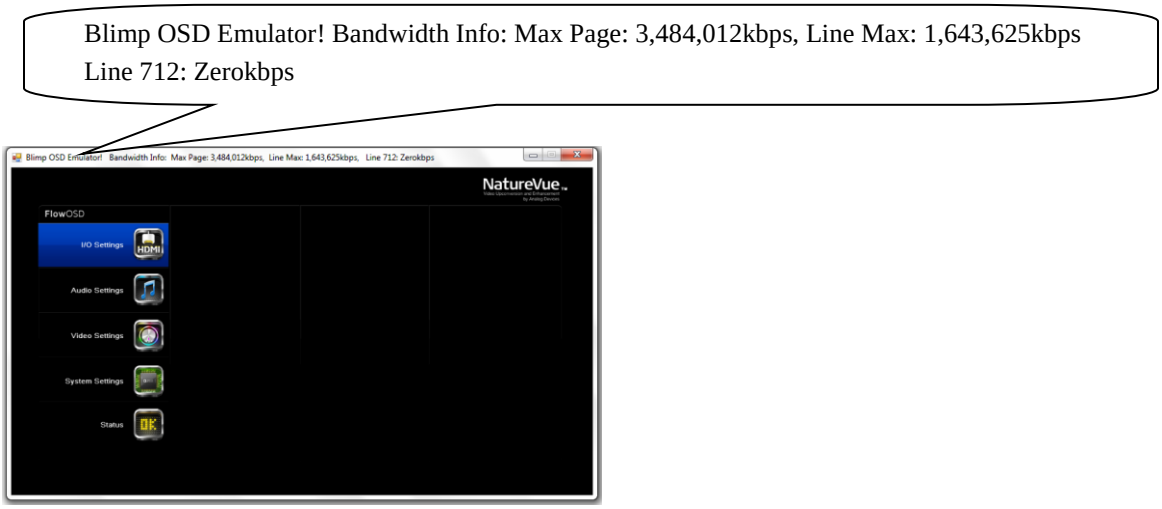


Figure 61: DDR2 BW information

A Standard resolution can be selected from the ComboBox in emulator window and this will affect the emulator size accordingly (as shown in belowFigure 62).

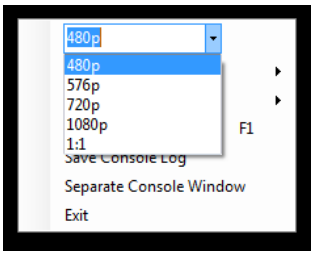


Figure 62: Resolution selection

Bandwidth Config will have format and resolution selection as shown inFigure 63, where the user can change and see the DDR2 BW warning. Depends on the formats and resolution selection the color of each line can be changed as like below:

- If the calculated line max is <80% of threshold value – shows green color bar
- If the calculated line max is between 80 - 90% of threshold value – shows yellow color bar
- If the calculated line max is >90% of threshold value – shows red color bar

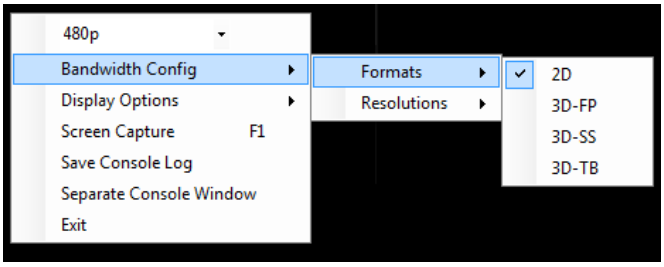


Figure 63: DDR2 BW configuration

By pressing “F6” key the colored bar will be displayed to show the DDR2 bandwidth for each line.

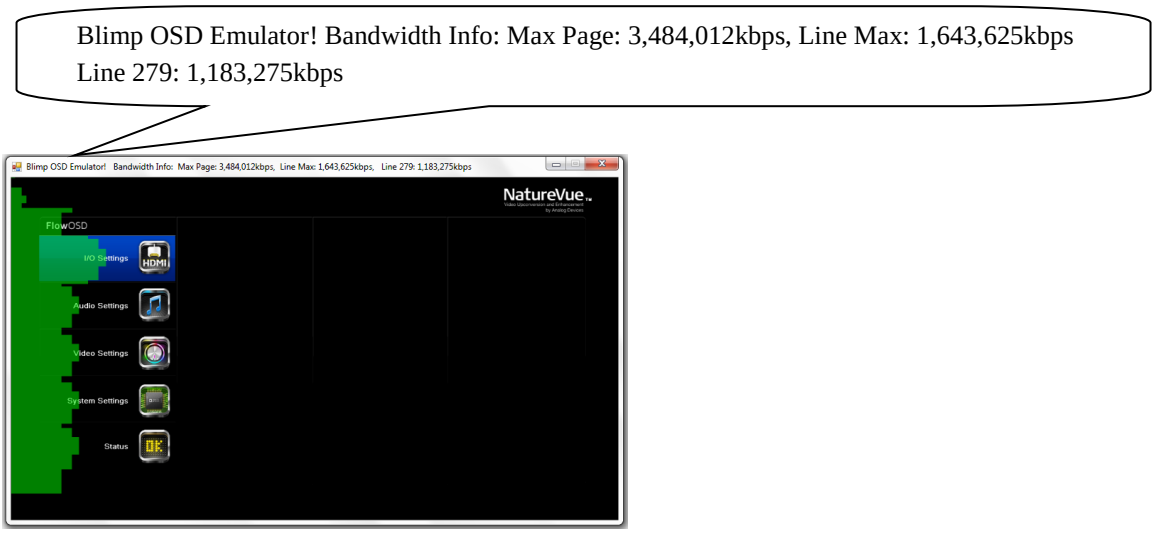


Figure 64: DDR2 BW information for each line using colored bar

In Display options, if the BW reference lines are checked then vertical dotted line will be shown as reference.

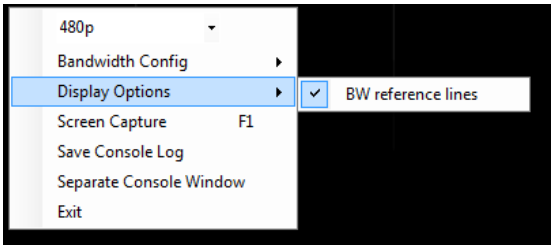


Figure 65: Display option selection

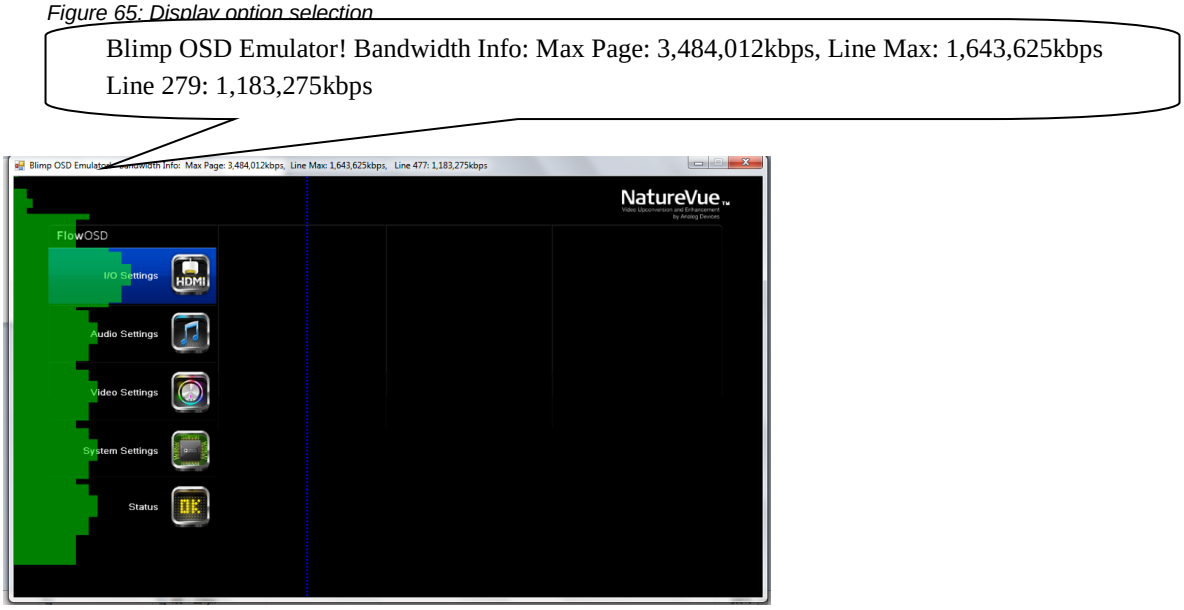


Figure 66: DDR2 BW reference vertical blue color dotted line.

18.4.1 DDR2 Bandwidth Limit for OSD

ADV800x supports industrial standard Double Data Rate (DDR2) SDRAM from 256Mbit to 2Gbit device sizes. The ADV800X uses DDR2 memory to enable the de-interlacer, scaler and OSD features. DDR2 usage estimates are on the blocks that use the DDR2, i.e. Primary VSP (PVSP), Secondary VSP (SVSP) & OSD. DDR2 bandwidth is usually expressed in units of bytes/second. The bandwidth would increase when either video timing or input video resolution is higher.

The DDR2 efficiency has been measured by keeping only the OSD features and by passed the VSP Application that uses either PVSP or SVSP or both, means VSP should not use any bandwidth and so OSD would have full bus access. In this way DDR2 efficiency for OSD are measured by doing test experiments.

The test results are done first with more accurate estimate of DDR2 usage for 2D and 3D video timing when using emulator bandwidth value.

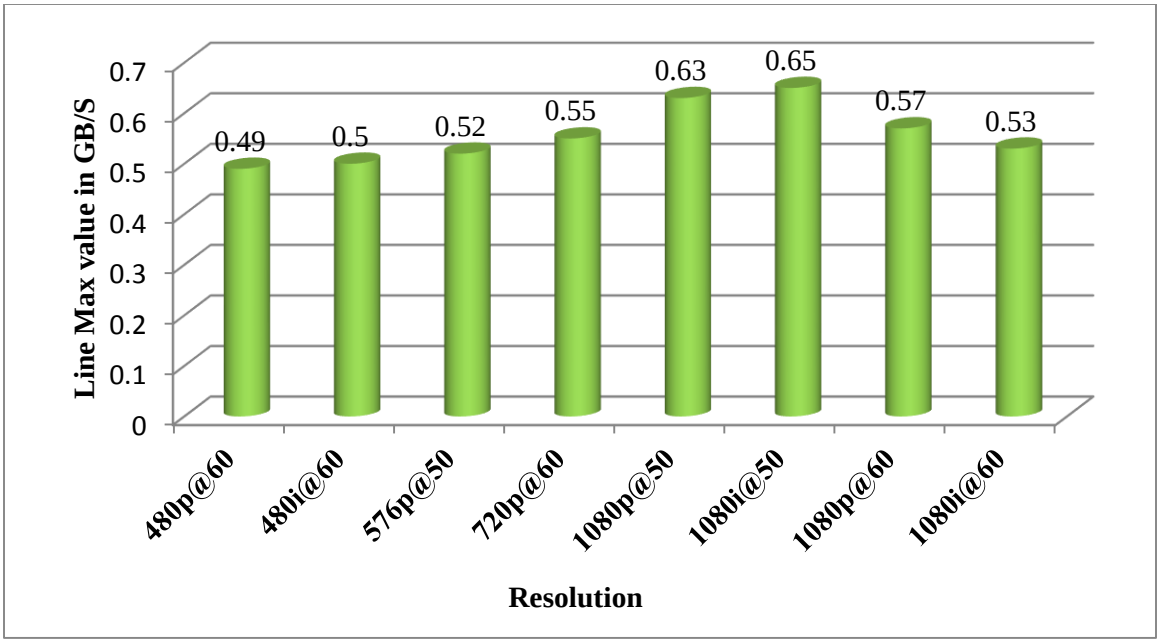


Figure 67: Bandwidth limit for 2D mode.

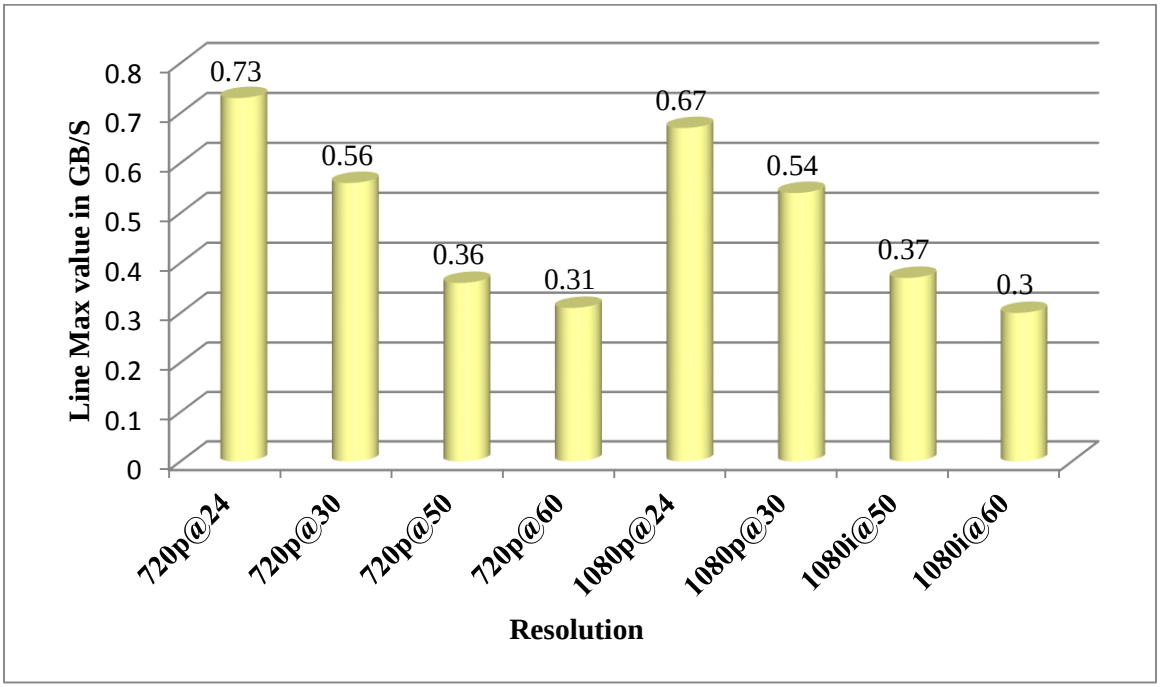
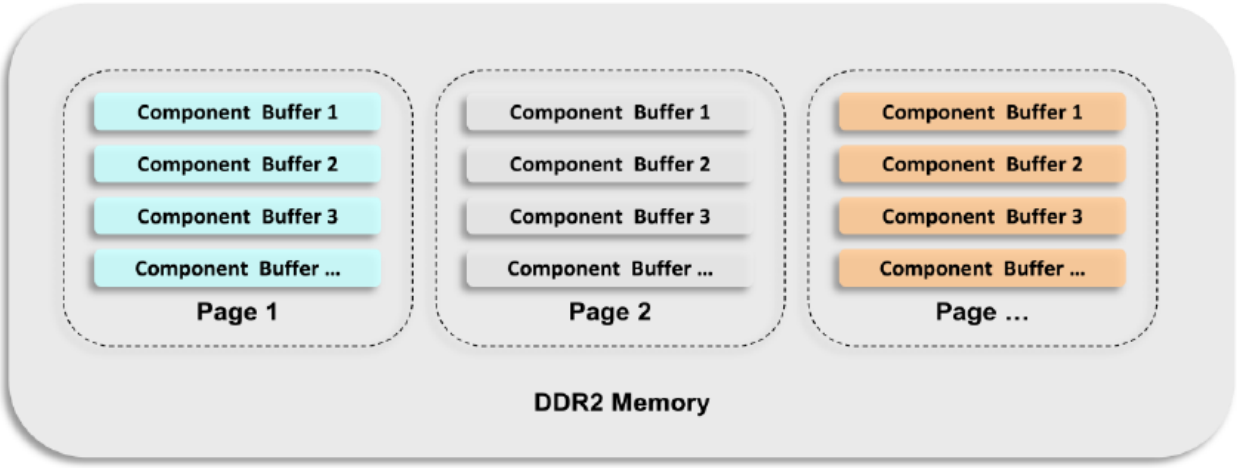


Figure 68: Bandwidth limit for 3D mode – frame packing.

19 Static and Dynamic Page Buffer

Blimp will support both static and dynamic page buffering technique.

In Static page buffering technique all “component buffers” are static memory and will never be released, which is the root cause that is so big in DDR2 memory.As all components have been drawn, it is no need to draw them again when displaying. This gives fastest OSD response.



Blimp will automatically switch to Dynamic page buffering, when the DDR2 size is not enough to store all the pages.

In Dynamic page buffering, there is only one thing that needs to customize:

Total DDR2 Size for page buffers (t_ddr2_page_size)

Firmware code will calculate maximum page size, then use this size (m_page_size) as the size of each page buffer.

Total buffer number = t_ddr2_page_size / m_page_size

By using upper equation, firmware would be able to calculate the page buffer number. If this number is 0, firmware won’t stop but nothing will be displayed.

All these information will be output as log information. So user will know how big each buffer is and how many buffers the firmware code is using.

If firmware allocates page buffer, draw and display just before going to show a page, the OSD response may be some slow. Here we introduce “pre-fetch” technique.

It’s very hard for Blimp or firmware code to determine page flow. So here firmware and Blimp allow user to pre-fetch a page into page buffer manually by calling one API function.

For example:

```
ADIAPI_OSDEgPrefetchPage (pageManager.main);
```

Here first empty page buffer will be detected. If not found, free oldest hidden one. Allocate found page buffer to “pageManager.main”. Draw “pageManager.main” when OSD is in idle state.

Note:

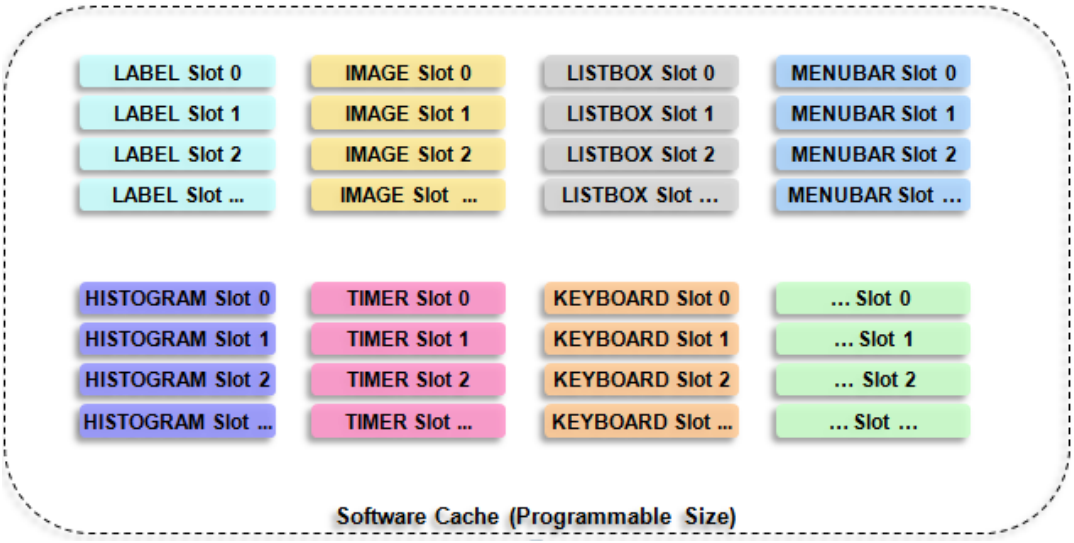
1. More than one page could be pre-fetched.
2. Pre-fetching will only succeed when there is “available” page buffer.
3. Even if step 2 failed, firmware code will try to allocate buffer again before displaying this page.
4. If step 3 also failed, this page won’t be displayed.

20 Cache

The software cache in firmware is a standalone software module, which includes the following features:

- Use a static buffer with programmable size as cache buffer.
- First-in and First-out scheme
- Return unique handle for each created component.
- Fetch component data from DDR2 memory according to the handle.
- Support to check dirty components. If the component has no change it will not be written back to DDR2 memory.

- Classified buffers for different kinds of components.



notes:

- Each slot is 4-byte aligned.
- The size of each kind of slot is corresponding to the component structure.
- Each slot type is corresponding to a component class.
- Currently there are 20 kinds of slots.
- The total size of software cache is programmable.

When creating a component, both region and component structures are allocated in cache. And the Maximum cache slot will be configurable in the project setting.

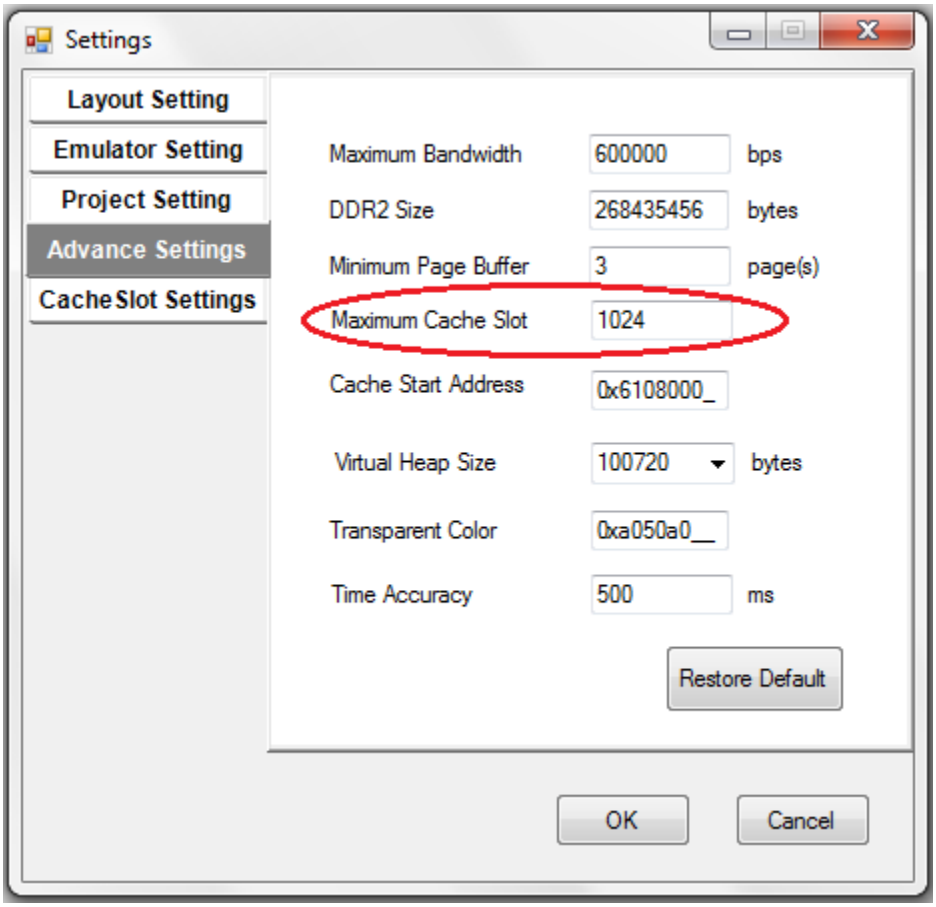


Figure 69: Max Cache slot setting in Project setting window.

Appendix A: Font & Image Library

Graphic engine uses library to manage different fonts and all icons & images. Both font library and image library must conform to specific format:

Font Library:

Table 51: Font Library Format

	Byte 0	**Byte 1**	**Byte2**	**Byte 3 … N[1]**
Character 0	Left Margin	Right Margin	Character Width	Character Data
Character 1	Left Margin	Right Margin	Character Width	Character Data
…	…	…	…	…
Character M	Left Margin	Right Margin	Character Width	Character Data

Image Library

Table 52: Image Library

	Byte 0~1	**Byte 2~3**	**Byte 4~5**	**Byte 6~7**	**Byte 8~9**	**Byte 10 ~11**	**Byte 12 …**
Image 0	Image Width	Image Height	Image Active Width	Image Left Margin	Image Right Margin	Image Color Depth	Image Data
Image 1	Image Width	Image Height	Image Active Width	Image Left Margin	Image Right Margin	Image Color Depth	Image Data
…	…	…	…				…
Image	Image	Image	Image	Image	Image	Image	Image

M	Width	Height	Active Depth	Left Margin	RightMargin	Color Depth	Data

Besides, there is also an image address lookup table for DMA to find different image. It has the following format:

Table 53: Image Lookup Table

	Byte 0 ~ 3
Image 0	Address in DDR2 Memory
Image 1	Address in DDR2 Memory
…	…
Image M	Address in DDR2 Memory

Appendix B: HARDWARE DRIVER

OSD Driver

OSD driver is for configuring OSD registers and region RAM to display OSD GUI. Table 51 lists the open API functions in this driver. More details can be found in Appendix C.

Table 54: OSD Driver API

API	**Descripton**
ADIAPI_OSDInitCore	Initialize OSD Core
ADIAPI_OSDSetCoreOutRes	Set output resolution of OSD Core
ADIAPI_OSDConfigPlane1	Configure plane properties
ADIAPI_OSDSetPlanePosition	Set plane position
ADIAPI_OSDSetPlaneSize	Set plane size
ADIAPI_OSDSetPlaneWeight	Set plane alpha value
ADIAPI_OSDInitCop	Initialize OSD Co-processor
ADIAPI_OSDConfigCopRgnRam	Configure region ram in OSD Co-processor
ADIAPI_OSDUpdateRgnAddr	Update region address in DDR2 memory
ADIAPI_OSDUpdateRgnSize	Set region size

ADIAPI_OSDUpdateRgnPosition	Set region location
ADIAPI_OSDUpdateRgnEn	Set region enable or disable
ADIAPI_OSDUpdateRgnOpqEn	Set region opaque or transparent
ADIAPI_OSDUpdateRgnEffect	Set region effect
ADIAPI_OSDUpdateRgnAniIndex	Set region corresponding animation
ADIAPI_OSDConfigAnimation	Configure animation parameter
ADIAPI_OSDUpdateAniEn	Enable or disable animation
ADIAPI_OSDUpdateAniStep	Update animation step value
ADIAPI_OSDUpdateAniType	Update animation type
ADIAPI_OSDSetExtOSDBufAddr	Configure buffer address for external input OSD
ADIAPI_OSDSetExtOSDSize	Set external input OSD size
ADIAPI_OSDSetResolution	Configure OSD size and scalar output resolution

Timer Driver

There are 8 timers which is 32-bits in OSD module to support accurate delay, which reduces hardware requirement for host CPU. Firmware has implemented software timer by just using only one hardware timer, so user application has no timer number limit although there are only 8 hardware timers.

For software timer, refer to 2.2.3 Classes in Graphic Engine.

Table 52 lists the open API functions of timer driver.

Table 55: Timer Driver API

API	Descripton
ADIAPI_TimerInit	**Initialize hardware timer and set timer to run**