APPLICATION NOTES

4D SYSTEMS

TURNING TECHNOLOGY INTO ART

ViSi: Diablo16 Inherent Widgets

DOCUMENT DATE:	2nd JULY 2020
DOCUMENT REVISION:	1.0

WWW.4DSYSTEMS.COM.AU

Description

This application note provides instructions for designing and using the Diablo16 Inherent widgets for a ViSi application.

Before getting started, the following are required:

Hardware

- Any 4D Systems display module powered by any of the following processors:
    - Diablo16 (with PmmC version 2.2 or higher)
- Programming Adaptor for target display module

Software

- Workshop4 IDE

This application note comes with one (1) ViSi project:

- Diablo_Inherents.4DViSi

Note: Using a non-4D programming interface could damage the processor and void the warranty.

Content

Application Overview

This document is mainly focused on showing the simple use of the Diablo16 Inherent widgets on the ViSi environment of the Workshop4 IDE. This type of widget is very useful in generating Graphical User Interfaces without the need for a uSD card. The parameters of each widget of this type resides in the user code as data blocks. Therefore, each widget adds a small amount to the program/code size.

In Diablo16, Inherent Widget resources are stored in a FlashBank. Workshop4 utilizes FLASHBANK_5 by default.

These widgets can be designed and manipulated using the properties tab in the Object Inspector of the ViSi environment. For more details, please refer to the following manual:

- **Workshop4 Widgets Reference Manual**

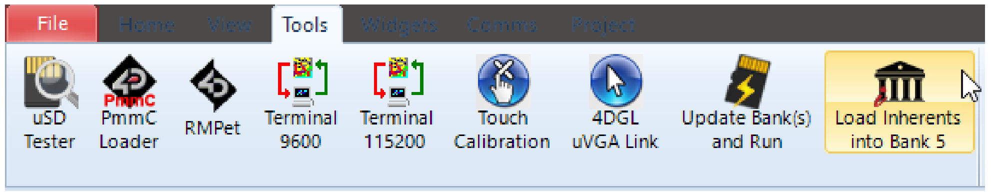
The simple project developed in this application note demonstrates an Inherent Switch Widget acting as a toggle switch for controlling an Inherent Media Led widget.

Setup Procedure

For instructions on how to launch Workshop4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note:

- **ViSi Getting Started - First Project for Picaso and Diablo16**

Workshop4 utilizes FLASHBANK_5 by default. The inherent widgets resources need to be uploaded using the Workshop4 tool.



Ensure that the Diablo display module is connected to the selected port and run the tool to upload the inherent widget resources.

Create a New Project

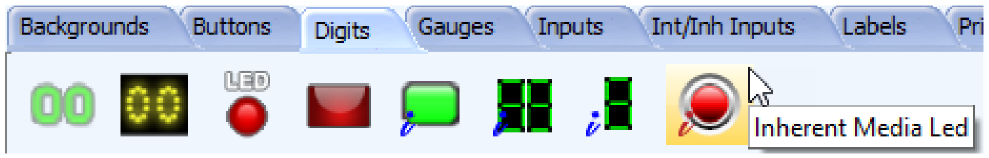
For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note:

- **ViSi Getting Started - First Project for Picaso and Diablo16**

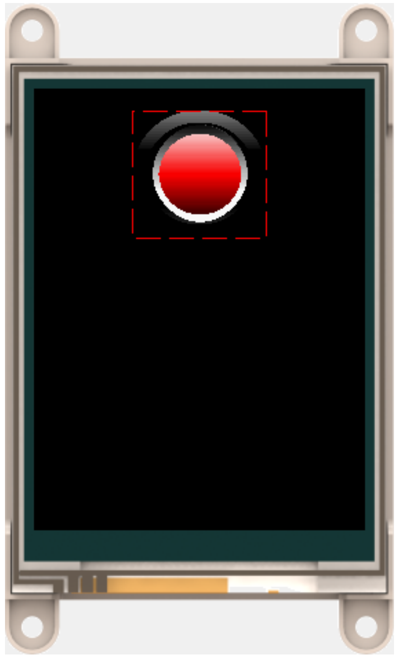
Design the Project

Adding an Inherent Media Led

Add the widget to the Form by clicking on the Digits tab, then selecting the icon as shown below.



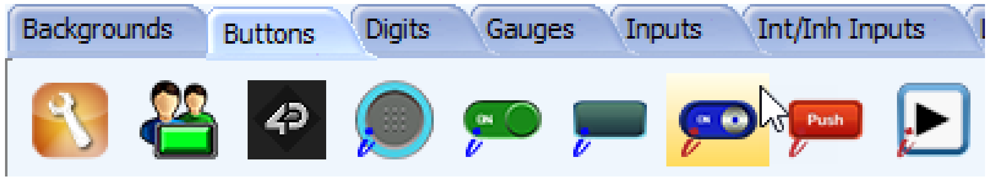
Click the WYSIWYG screen to place it, then drag it to the desired position.



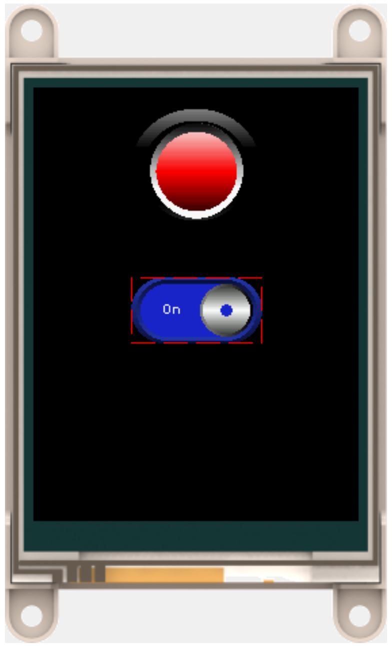
The Led widget can be modified as needed through the **Object Inspector**.

Adding an Inherent Switch

Add the widget to the Form by clicking on the Buttons tab, then selecting the icon as shown below.



Click the WYSIWYG screen to place it, then drag it to the desired position.



The Switch widget can be modified as needed through the **Object Inspector**.

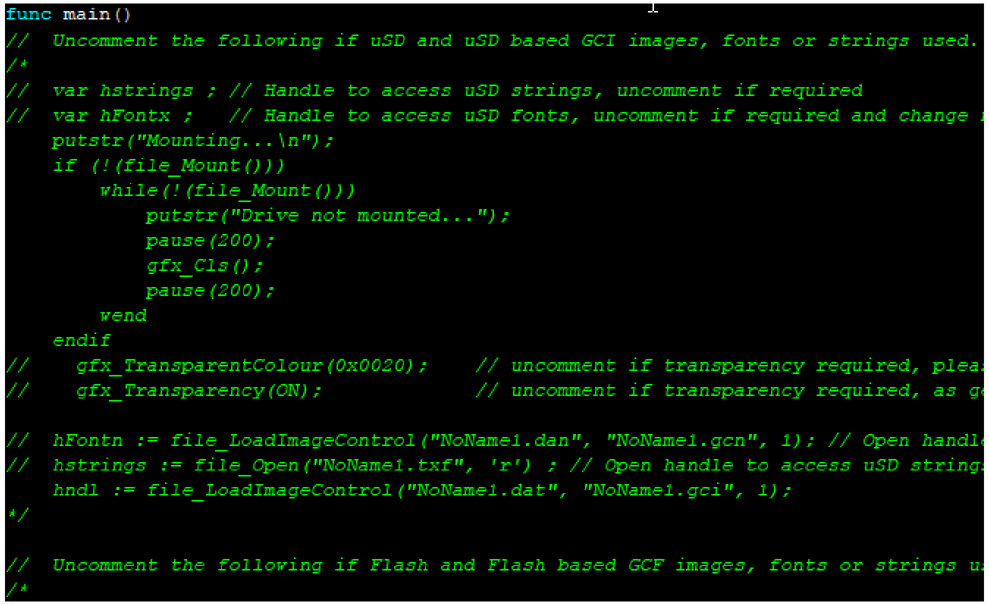
Programming the Display

It is always ideal to prepare the graphics interface before moving to programming the display. This lessens the number of times the users will have to proceed on doing a **Paste Code** action in order to regenerate the widget parameters in the code editor.

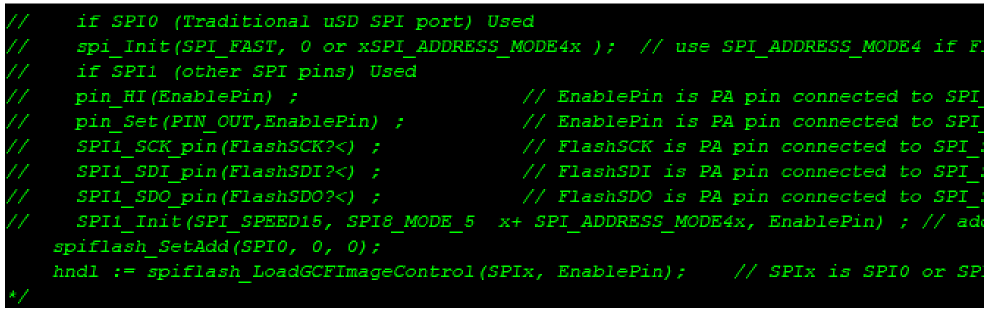
When the user interface is near its final stages, it is the best time to proceed with programming.

Initial ViSi Code

When starting a ViSi project, it includes a bare minimum project that contains commented initialization code for connected storage device.

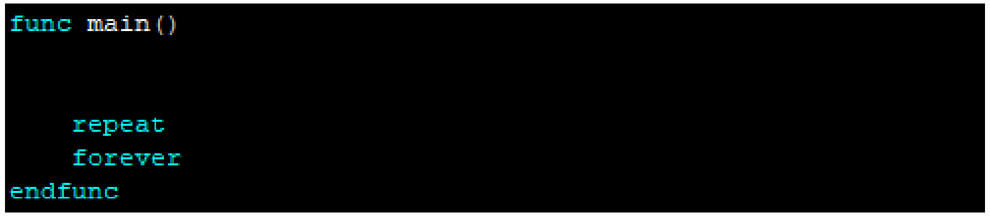


```
func main()
//  Uncomment the following if uSD and uSD based GCI images, fonts or strings used.
/*
//  var hstrings ; // Handle to access uSD strings, uncomment if required
//  var hFontx ;   // Handle to access uSD fonts, uncomment if required and change
    putstr("Mounting...\n");
    if (!(file_Mount()))
        while(!(file_Mount()))
            putstr("Drive not mounted...");
            pause(200);
            gfx_Cls();
            pause(200);
        wend
    endif
//    gfx_TransparentColour(0x0020);    // uncomment if transparency required, plea
//    gfx_Transparency(ON);             // uncomment if transparency required, as g

//  hFontn := file_LoadImageControl("NoName1.dan", "NoName1.gcn", 1); // Open handl
//  hstrings := file_Open("NoName1.txf", 'r') ; // Open handle to access uSD string
    hndl := file_LoadImageControl("NoName1.dat", "NoName1.gci", 1);
*/

//  Uncomment the following if Flash and Flash based GCF images, fonts or strings u
/*
```



```
//    if SPI0 (Traditional uSD SPI port) Used
//    spi_Init(SPI_FAST, 0 or xSPI_ADDRESS_MODE4x );  // use SPI_ADDRESS_MODE4 if F
//    if SPI1 (other SPI pins) Used
//    pin_HI(EnablePin) ;                   // EnablePin is PA pin connected to SPI_
//    pin_Set(PIN_OUT,EnablePin) ;          // EnablePin is PA pin connected to SPI_
//    SPI1_SCK_pin(FlashSCK?<) ;            // FlashSCK is PA pin connected to SPI_
//    SPI1_SDI_pin(FlashSDI?<) ;            // FlashSDI is PA pin connected to SPI_
//    SPI1_SDO_pin(FlashSDO?<) ;            // FlashSDO is PA pin connected to SPI_
//    SPI1_Init(SPI_SPEED15, SPI8_MODE_5  x+ SPI_ADDRESS_MODE4x, EnablePin) ; // ad
    spiflash_SetAdd(SPI0, 0, 0);
    hndl := spiflash_LoadGCFImageControl(SPIx, EnablePin);    // SPIx is SPI0 or SP
*/
```

Since Inherent widgets utilizes a FlashBank of a Diablo16 display module, the project doesn't require any storage device setup.



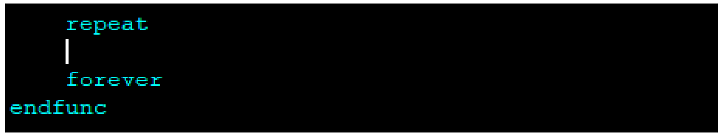
```
func main()


    repeat
    forever
endfunc
```

Feel free to remove the other unnecessary lines of code as done in this application note project.

Paste Inherent Led Code

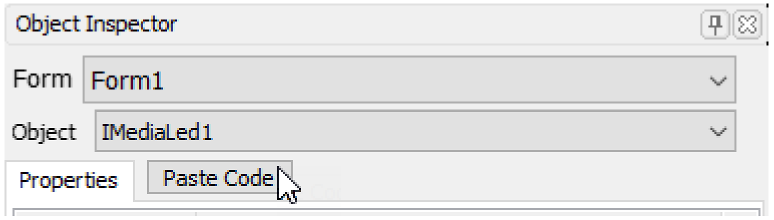
Like the GCI widgets, ViSi provides Paste Code feature for Inherent widgets.

Place the blinking cursor inside the repeat-forever loop.



```
    repeat
    |
    forever
endfunc
```

Go to the Object Inspector, choose IMediaLed1 and click **Paste Code**



This will paste the function and initialization code for rendering the Media Led widget.

```
// IMediaLed1 1.0 generated 6/29/2020 11:04:06 PM
vIMediaLed1RAM[WIDGET_TAG] := gradientRAM ;              // uncomment gradientRAM variable and s
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaButton, state, vIMediaLed1RAM, IIMediaLed1, 19, 0) ;
```

The second line is for widget initialization. This line, as described in the comments, is only required to be set once. Move this line before the repeat forever loop.

```
func main()

    vIMediaLed1RAM[WIDGET_TAG] := gradientRAM ;            // uncomment grad

    repeat

    // IMediaLed1 1.0 generated 6/29/2020 11:04:06 PM
    flash_FunctionCall(FLASHBANK_5, Fgfx_MediaButton, state, vIMediaLed1RAM,

    forever
endfunc
```

Additionally, the widget needs to be drawn initially on the screen.

```
vIMediaLed1RAM[WIDGET_TAG] := gradientRAM ;            // uncomment grad
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaButton, state, vIMediaLed1RAM,
```

Initially, we'd want to draw it in its OFF (0) state.

```
vIMediaLed1RAM[WIDGET_TAG] := gradientRAM ;             // uncomment gradie
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaButton, 0, vIMediaLed1RAM, IIMed
```

Before moving forward, note the comment on the line:

```
vIMediaLed1RAM[WIDGET_TAG] := gradientRAM ;
```

The comment reads:

```
// uncomment gradientRAM variable and set to a minimum of 50
```

On the top of the code, you'll notice the array "**gradientRAM**" commented out. Uncomment this and replace xxx with 50.

```
//var gradientRAM[29+xxx*2] := [-1,-1,-9999,0,0,xxx] ;  // uncomment and replace xxx with
var gradientRAM[129] := [-1,-1,-9999,0,0,50]; // xxx = 50, min requirement for IMediaLed1
```

Note: xxx should be equal to the maximum value required for all "media" widget

Another comment reads:

```
requires a minimum #STACK of 280
```

As advised in the comments, adjust the stack size to the maximum value required for all widgets.

```
#STACK 280

//var gradientRAM[29+xxx*2] := [-1,-1,-9999,0,0,xxx] ;  // uncomment and replace xxx with
var gradientRAM[129] := [-1,-1,-9999,0,0,50]; // xxx = 50, min requirement for IMediaLed1
```

The parameters and variables required by the function are also automatically inserted near the start of the code.

```
#DATA
    // IMediaLed1 Data Start
    word IIMediaLed1 71, 16, 98, 93, 0x9B18, 0, BLACK, SILVER, BLACK, DARKRED,
        RED, 0, BLUE, 0x0, CIMediaLed1, 2, 1
    byte CIMediaLed1 "Text",0
    // IMediaLed1 Data End
#END
var vIMediaLed1RAM[WIDGET_RAM_SPACE] ;
```

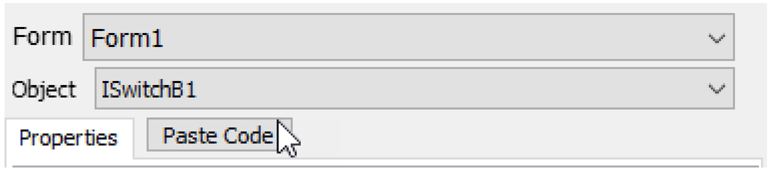
Notice the fixed integers and colour included in the data block for IMediaLed1. These parameters are based on the values in the Object Inspector and may have to be regenerated by doing a new paste code action if the widget parameters are changed.

Paste the Internal Switch Code

Place the blinking cursor in the place where you want the code is pasted.

```
repeat

// IMediaLed1 1.0 generated 6/29/2020 11:04:06 PM
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaButton, state, vIMediaLed1RAM, IIMediaLed1, 19, 0) ;



forever
```

Go to the Object Inspector, choose ISwitchB1 and click **Paste Code**


Form Form1
Object ISwitchB1
Properties | Paste Code

This will paste the function for rendering the Switch widget.

```
// ISwitchB1 1.0 generated 6/29/2020 11:12:18 PM
flash_FunctionCall(FLASHBANK_5, Fgfx_Switch2, state, vISwitchB1RAM, IISwitchB1, 17, 0x18) ;
```

The parameters and variables required by the function are also automatically added to the data block near the start of the code.

```
#DATA
    // ISwitchB1 Data Start
    word IISwitchB1 72, 140, 97, 49, 0x0, 0x21B5, 0x0887, 2, 3, 0x1939, 0x31D0,
        CnISwitchB1, CfISwitchB1, 2, 1, WHITE, 0x0866
    byte CnISwitchB1 "On",0
    byte CfISwitchB1 "Off",0
    // ISwitchB1 Data End
    // IMediaLed1 Data Start
    word IIMediaLed1 71, 16, 98, 93, 0x9B18, 0, BLACK, SILVER, BLACK, DARKRED,
        RED, 0, BLUE, 0x0, CIMediaLed1, 2, 1
    byte CIMediaLed1 "Text",0
    // IMediaLed1 Data End
#END
var vISwitchB1RAM[WIDGET_RAM_SPACE] ;
var vIMediaLed1RAM[WIDGET_RAM_SPACE] ;
```

Notice the fixed integers and colour included in the data block for the ISwitchB1. These parameters are based on the values in the Object Inspector and may have to be regenerated by doing a new paste code action if the widget parameters are changed.

Widget Handling

The best way to manage multiple widgets is to setup a widget handler. The function **widget_Create** generates a widget control capable of holding count elements and returns a handle (pointer to the memory allocation) to the image control list. This handle will be used to access and display objects, as will be shown later.

```
var touchState;
var state, value, n;

ihndl := widget_Create(1); // Generate widget control handle
```

With the handle established, the widgets can then be added into the handle through the **widget_Add** function. This will add the Switch widget entry as index 0.

```
widget_Add(ihndl, 0, vISwitchB1RAM); // Add switch widget to handle
```

The function **widget_SetAttributes** is then used to enable the touch detection attribute on the button widget.

```
widget_SetAttributes(ihndl, 0, WIDGET_F_TOUCH_ENABLE);
```

Touch Detection

The touchscreen for the display module is enabled through the **touch_Set** function.

```
touch_Set(TOUCH_ENABLE); // Enable touch
```

In the repeat loop, the touchscreen status is constantly checked for changes.

```
repeat
    touchState := touch_Get(TOUCH_STATUS); // Get Touch Status
```

If a touch press is detected, the conditional proceeds to check which widget is touched through the function **widget_Touched**. This function will scan all the widgets in the handle with enabled touch detection attribute.

```
if (touchState == TOUCH_PRESSED)
    n := widget_Touched(ihndl, ALL); // Check if any widget was touched
```

If the touch function returns index 0, this means that the switch is pressed. The switch is then checked if it was pressed on the left or right side. If the switch is pressed in the right, the new state should be ON (1). Otherwise, new state should be OFF (0).

```
if (n == 0) // Switch is touched
    // check if the switch is pressed in left or right
    state := (touch_Get(TOUCH_GETX) > switchMidX) ? 1 : 0;

    // IMediaLed1 1.0 generated 6/29/2020 1:28:49 AM
    flash_FunctionCall(FLASHBANK_5, Fgfx_MediaButton, state, vIMediaLed1RAM, IIMediaLed1, 19, 0) ;

    // ISwitchB1 1.0 generated 6/29/2020 1:31:22 AM
    flash_FunctionCall(FLASHBANK_5, Fgfx_Switch2, state, vISwitchB1RAM, IISwitchB1, 17, 0x18) ;
endif
```

Both the Switch and the LED are then updated accordingly.

To get this to work, the middle X coordinate of the switch needs to be computed at least once.

```
switchMidX := widget_GetWord(ihndl, 0, WIDGET_XPOS) +
    (widget_GetWord(ihndl, 0, WIDGET_WIDTH) / 2);
```

Run the Program

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section “**Run the Program**” of the application note:

- **ViSi Getting Started - First Project for Picaso and Diablo16**

Proprietary Information

Disclaimer of Warranties & Limitation of Liability