APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi: Diablo Inherent Input Widgets

DOCUMENT DATE:	**2nd JULY 2020**
DOCUMENT REVISION:	**1.0**

W W W . 4 D S Y S T E M S . C O M . A U

Description

This application note provides instructions for designing and using the Inherent knob and slider 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

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

- Diablo_Inherent_Inputs.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 Inherent knob and slider 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 microSD 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**

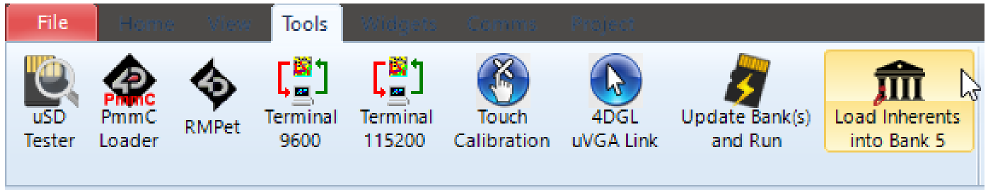
The simple project developed in this application note demonstrates an Inherent Knob and Inherent Slider used to update an Internal LedDigits.

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

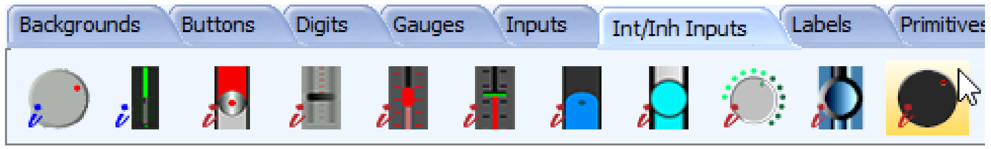
For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note:

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

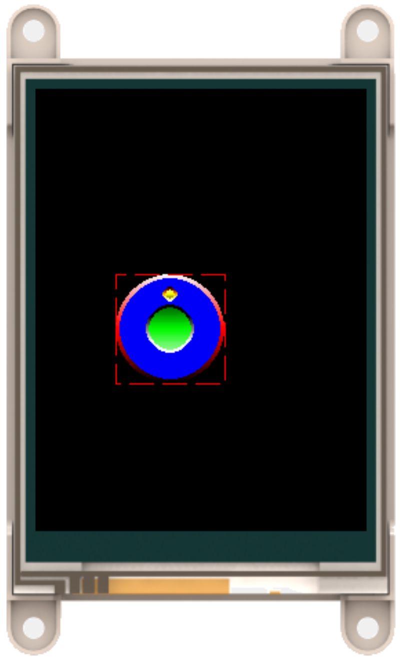
Design the Project

Adding an Inherent Knob

Add the widget to the Form by clicking on the **Int/Inh Inputs** tab, then selecting the icon as shown below.



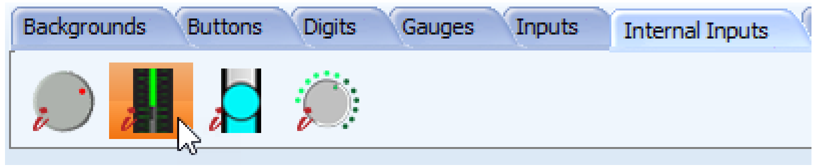
Click the WYSIWYG screen to place it, then drag it to the desired position.



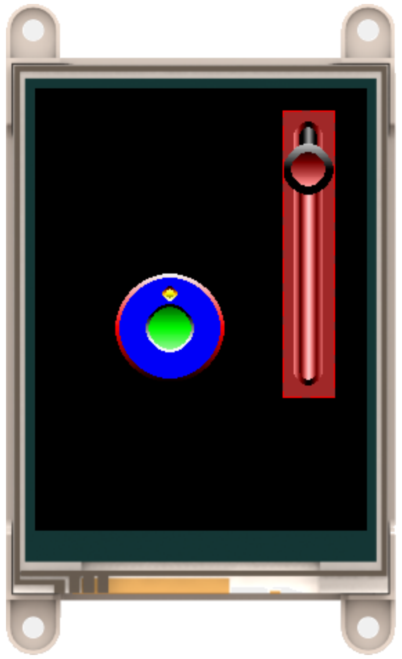
The Knob widget can be modified as needed through the **Object Inspector**.

Adding an Inherent Slider

Add the widget to the Form by clicking on the **Int/Inh Inputs** tab, then selecting the icon as shown below.



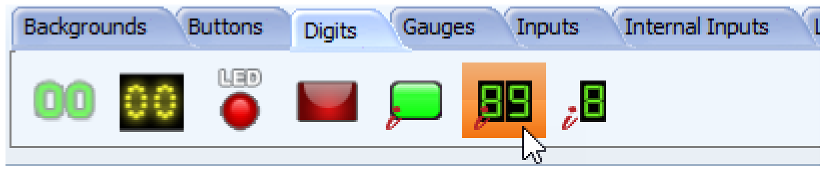
Click the WYSIWYG screen to place it, then drag it to the desired position.



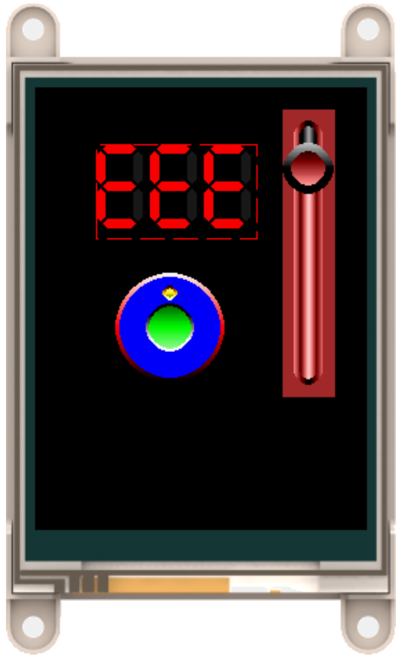
The Slider widget can be modified as needed through the **Object Inspector**.

Adding a Internal LedDigits

Add an Internal LedDigits to the Form by clicking on the Digits tab, then selecting the Internal LedDigits as shown below.



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



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

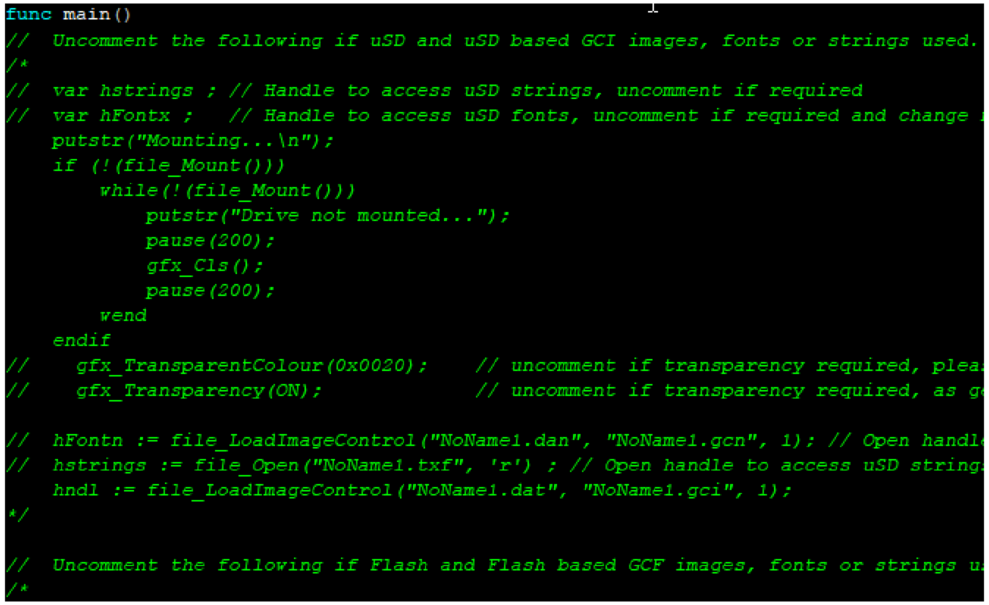
Programming the Display

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.

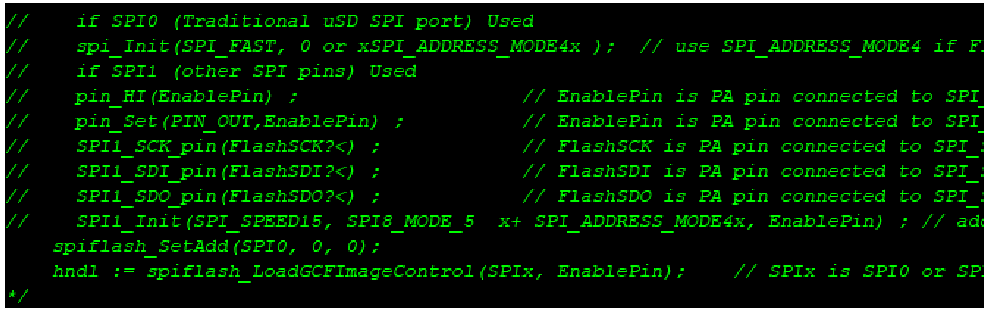
When the user interface is near its final stages, it is the best time to proceed with programming.

Initial ViSi Code

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

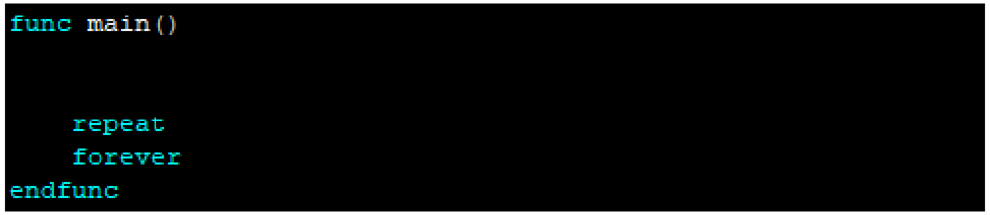


```
func main()
//  Uncomment the following if uSD and uSD based GCI images, fonts or strings used.
/*
//  var hstrings ; // Handle to access uSD strings, uncomment if required
//  var hFontx ;   // Handle to access uSD fonts, uncomment if required and change 
    putstr("Mounting...\n");
    if (!(file_Mount()))
        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.

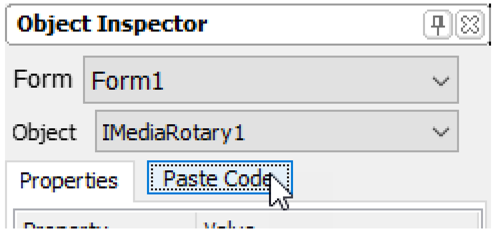
Paste the Inherent Knob Code

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 IMediaRotary1 and click **Paste Code**



This will paste the useful lines of code that can be used for handling the know widget.

```
// IMediaRotary1 1.0 generated 7/1/2020 10:19:58 PM
vIMediaRotary1RAM[WIDGET_TAG] := gradientRAM ;           // uncomment gradientRAM variable and set
posn := gfx_XYrotToVal(x-98, y-174, XYROT_SOUTH, 45, 315, 0, 16);
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaRotary, posn, vIMediaRotary1RAM, IIMediaRotary1, 19, 0) ;
```

The **first** line is simply a comment of containing the widget name and the date and time the piece of code was generated.

The **second** line can be used during setup and should be set at least once.

The **third** line can be used to compute for the new value (*posn*) of the knob given the touch position (*x, y*).

The **last** line can be used to update the widget value displayed on the screen by assigning a new value (*posn*).

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

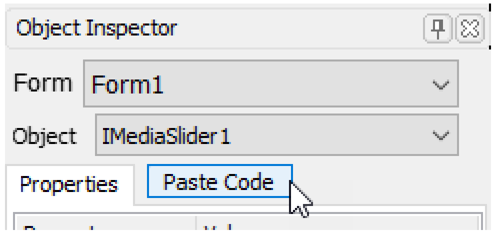
```
// IMediaRotary1 Data Start
word IIMediaRotary1 58, 134, 80, 80, 16, RED, BLUE, GRAY, LIME, ORANGE,
    YELLOW, 0x30FE, 3, 25, 13, 36, 2, 135, 405, 21
// IMediaRotary1 Data End
```

Notice the fixed integers and colour included in the data block and the lines of code for the knob.

These parameters in the data block and the useful lines of code discussed above 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 Slider Code

Place the blinking cursor in the place where you want the code to be pasted. Then go to the Object Inspector, choose IMediaSliderE1 and click **Paste Code**



This will paste the function for rendering the Slider widget.

```
// IMediaSlider1 1.0 generated 7/1/2020 10:21:34 PM
vIMediaSlider1RAM[WIDGET_TAG] := gradientRAM ;          // uncomment gradientRAM variable and set
posn := gfx_XYlinToVal(x, y, 0, 35, 204, 0, 100) ;
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaSlider, posn, vIMediaSlider1RAM, IIMediaSlider1, 16, 0) ;
```

The **first** line is simply a comment of containing the widget name and the date and time the piece of code was generated.

The **second** line can be used during setup and should be set at least once.

The **third** line can be used to compute for the new value (*posn*) of the slider given the touch position (*x, y*).

The **last** line can be used to update the widget value displayed on the screen by assigning a new value (*posn*).

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

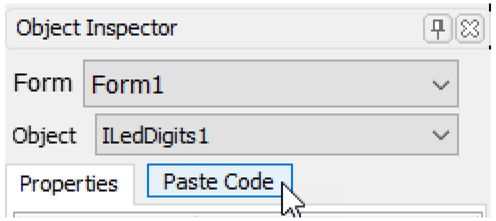
```
// IMediaSlider1 Data Start
word IIMediaSlider1 179, 16, 38, 208, 100, BROWN, BLACK, BROWN, BLACK,
    BROWN, BROWN, 0x1914, 22, 6, 4
// IMediaSlider1 Data End
```

Notice the fixed integers and colour included in the data block and the lines of code for the slider.

These parameters in the data block and the useful lines of code discussed above 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 LedDigits Code

Place the blinking cursor in the place where you want the code to be pasted. Then go to the Object Inspector, choose ILedDigits1 and click **Paste Code**



This will paste the function for rendering the LedDigits widget.

```
// ILedDigits1 1.0 generated 7/1/2020 8:17:21 PM
gfx_LedDigits(value, vILedDigits1RAM, IILedDigits1) ;
```

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

```
// ILedDigits1 Data Start
word IILedDigits1 44, 41, 117, 69, 3, 5, 0, 3, WHEAT, 0x18C3, 0x0
// ILedDigits1 Data End
```

Adding Necessary Variables

Some variables included in the generated code from Workshop4's Paste Code feature needs to be added to the project.

```
var ihndl;
var touchState, n;
var posn, x, y;
```

Extra variables ihndl and touchState was added for widget and touch handling that will be discussed in the next sections.

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.

```
ihndl := widget_Create(2);
```

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

```
widget_Add(ihndl, 0, vIMediaRotary1RAM);
widget_Add(ihndl, 1, vIMediaSlider1RAM);
```

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

```
widget_SetAttributes(ihndl, 0, WIDGET_F_TOUCH_ENABLE);
widget_SetAttributes(ihndl, 1, WIDGET_F_TOUCH_ENABLE);
```

As discussed on the Paste Code sections, there are some lines generated that needs to be set at least once. These lines need to be moved before the repeat forever loop.

```
vIMediaRotary1RAM[WIDGET_TAG] := gradientRAM ;
vIMediaSlider1RAM[WIDGET_TAG] := gradientRAM ;
```

Checking the comments on these lines you'll notice the following instructions:

```
// uncomment gradientRAM variable and set to a minimum of 41,
// uncomment gradientRAM variable and set to a minimum of 19,
```

Follow the instruction to uncomment the gradientRAM variable.

```
var gradientRAM[29+xxx*2] := [-1,-1,-9999,0,0,xxx] ;
```

xxx needs to be replaced by the highest required value as discussed in the comments. In this case, the maximum value is 41.

```
var gradientRAM[111] := [-1,-1,-9999,0,0,41] ;
```

In order to show the widgets on the screen, their functions must be called at least once before the repeat-forever loop.

```
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaRotary, posn, vIMediaRotary1RAM, IIMediaRotary1, 19, 0) ;
flash_FunctionCall(FLASHBANK_5, Fgfx_MediaSlider, posn, vIMediaSlider1RAM, IIMediaSlider1, 16, 0) ;
gfx_LedDigits(0, vILedDigits1RAM, IILedDigits1) ;
```

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. The X and Y coordinates are also constantly stored for later calculations.

```
repeat
    touchState := touch_Get(TOUCH_STATUS);
    x := touch_Get(TOUCH_GETX);
    y := touch_Get(TOUCH_GETY);
```

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

```
if (touchState == TOUCH_PRESSED || touchState == TOUCH_MOVING)
    n := widget_Touched(ihndl, ALL); // Scan widget list looking for a touch
```

If the touch function returns index 0, this means that the knob is touched. The angular position is calculated using the **gfx_XYrotToval** function. The position is then used to update the knob widget and the LedDigits widget.

```
if (n == 0)
    posn := gfx_XYrotToVal(x-98, y-174, XYROT_SOUTH, 45, 315, 0, 16);
    flash_FunctionCall(FLASHBANK_5, Fgfx_MediaRotary, posn, vIMediaRotary1RAM
    gfx_LedDigits(posn, vILedDigits1RAM, IILedDigits1) ;
```

Otherwise if the touch function returns index 1, this means that the slider is touched. The linear position is calculated using the **gfx_XYlinToval** function. The position is then used to update the slider widget and the LedDigits widget.

```
else if (n == 1)
    posn := gfx_XYlinToVal(x, y, 0, 35, 204, 0, 100) ;
    flash_FunctionCall(FLASHBANK_5, Fgfx_MediaSlider, posn, vIMediaSlider1RAM
    gfx_LedDigits(posn, vILedDigits1RAM, IILedDigits1) ;
```

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