

ViSi: Diablo16 and Pixxi Internal (PmmC) Widgets

DOCUMENT DATE: 24th J DOCUMENT REVISION: 1.0

24th JUNE 2020



WWW.4DSYSTEMS.COM.AU

Description

This application note provides instructions for designing and using the Pixxi or Diablo16 Internal (PmmC) 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)
 - o Pixxi28/44
- Programming Adaptor for target display module

Software

- Workshop4 IDE

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

• PmmC_Widgets.4DViSi

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

Content

Description	2
Content	2
Application Overview	3
Setup Procedure	3
Create a New Project	3
Design the Project	4
Adding an Internal Led	4
Adding an Internal Button	4
Programming the Display	5
Paste Internal Led Code	5
Paste the Internal Button Code	6
Widget Handling	6
Touch Detection	7
Run the Program	7
Proprietary Information	8
Disclaimer of Warranties & Limitation of Liability	8

Application Overview

This document is mainly focused on showing the simple use of the PmmC widgets on the ViSi environment of the Workshop4 IDE. This type of widget is very useful in generating Graphical User Interfaces without the use of any external memory storage device. 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.

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 a Internal Button Widget acting as a momentary switch for controlling a Internal 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 any of the following application notes:

- ViSi Getting Started First Project for Picaso and Diablo16
- ViSi Getting Started First Project for Pixxi Display Modules

Create a New Project

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

- ViSi Getting Started First Project for Picaso and Diablo16
- ViSi Getting Started First Project for Pixxi Display Modules

Design the Project

Adding an Internal Led

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

Backgrounds	Buttons	Digits	Gauges Inputs Internal Inputs
00 00			, 88

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 Internal Button

Add a Internal Button to the Form by clicking on the Buttons tab, then selecting the Internal Button as shown below.



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



The Button 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.

Paste Internal Led Code

Similar to the GCI widgets, ViSi provides Paste Code feature for Internal widgets.

Place the blinking cursor inside the repeat-forever loop.



Go to the Object Inspector, choose ILed1 and click Paste Code

Object	Inspector		1 23
Form	Form1		~
Object	ILed1		~
Proper	ties Pas	te Code	
Prope	rty	Value	
Name		ILed1	
Alias		ILed1	

This will paste the function for rendering the Led widget.

```
// ILed1 1.0 generated 05/02/2020 11:41:48 AM
gfx_Led(value, ILed1RAM, IILed1) ;
```

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

#platform "pixxiLCD-35P4CT"

// Program Skeleton 1.5 generated 05/02/2020 9:26:25 AM

// #MODE RUNFLASH uncomment and set Destination to Flash to run from Flash, refer

#inherit "4DGL_16bitColours.fnc"

Notice the fixed integers and colour included in the data block for the ILed1 function gfx_Led. 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 Button Code

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



Go to the Object Inspector, choose IButtonD1 and click **Paste Code**

Object Inspector			
Form Form1		~	
Object IButton	01	~	
Properties Pa	aste Code		
Property	Value Value		
Name	IButtonD1		
Alias	IButtonD1		

This will paste the function for rendering the Button widget.



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



Notice the fixed integers and colour included in the data block for the IButtonD1 function gfx_Button4. 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.



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(hndl, 0, IButtonD1RAM); // Add Widget to handle

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

widget_SetAttributes(hndl, 0, WIDGET_F_TOUCH_ENABLE); // Enable touch for Button

Touch Detection

The touchscreen for the pixxiLCD display 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 of the widget in the handle with enabled touch detection attribute.

if(touchState == TOUCH_PRESSED)
 n := widget_Touched(hndl, ALL); //scan widget list, looking for a touch

If the touch function returns index 0, this means that the button is pressed. The button is updated to show its pressed state and the LED is turned on.



Similarly, when the touch is released over the button, the button is updated back to its normal state and LED is turned off.



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 any of the following application notes:

- ViSi Getting Started First Project for Picaso and Diablo16
- ViSi Getting Started First Project for Pixxi Display Modules

Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either expresses or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.