



ViSi: Internal LedDigits

DOCUMENT DATE: **25th JUNE 2020**
DOCUMENT REVISION: **1.0**

Description

This application note provides instructions for designing and using the Internal LedDigit and Internal LedDigits widget 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)
 - Pixxi28/44
- Programming Adaptor for target display module

Software

- Workshop4

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

- Internal_LedDigits.4DGenie

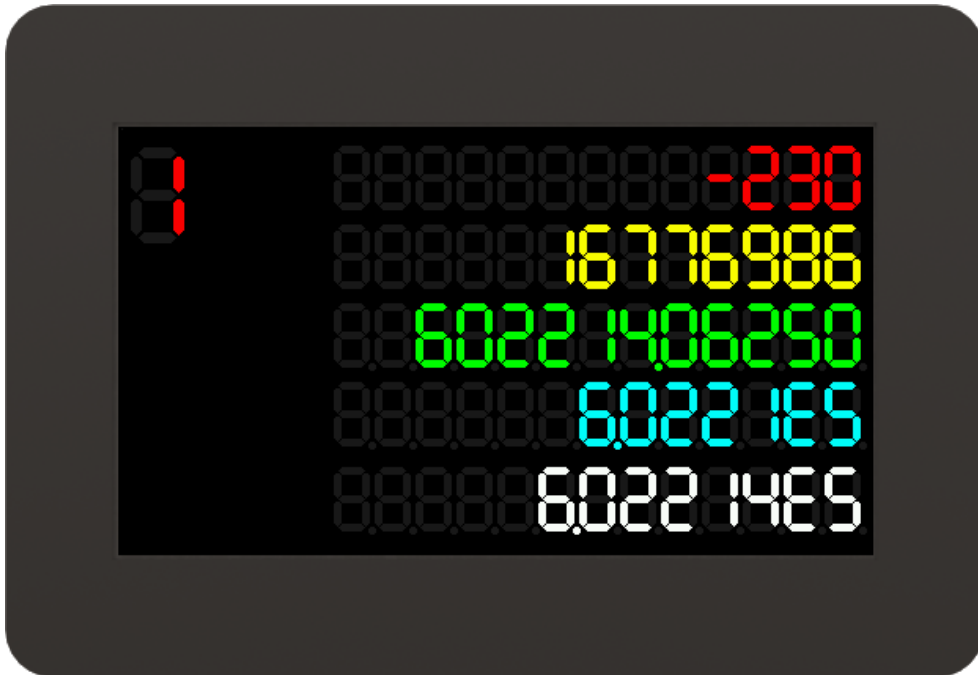
Note: Using a non-4D programming interface could damage the processor and void the warranty.

Content

Description	2
Content.....	2
Application Overview	3
Setup Procedure	3
Create a New Project	3
Design the Project.....	4
<i>Add Internal LedDigit</i>	<i>4</i>
<i>Add Internal LedDigits</i>	<i>5</i>
<i>Programming the Display</i>	<i>7</i>
Paste Internal LedDigit Code	7
Operating the LedDigit	7
Paste Internal LedDigits Codes	8
Displaying the LedDigits Value	9
Pause the Loop	10
Run the Program.....	10
Demonstration	10
Proprietary Information	11
Disclaimer of Warranties & Limitation of Liability	11

Application Overview

This document is mainly focused on showing the simple use of the Internal LedDigit and Internal LedDigits widget on the ViSi environment of the Workshop4 IDE. These widgets are very useful in displaying numerical values like temperature, volume, content percentage and many more. Depending on the user's preferences, these widgets can be manipulated using the properties tab in the Object Inspector of the ViSi environment.



The simple project developed in this application note demonstrates the various use and formats of the Internal LedDigit and 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 any of the following application notes:

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

Create a New Project

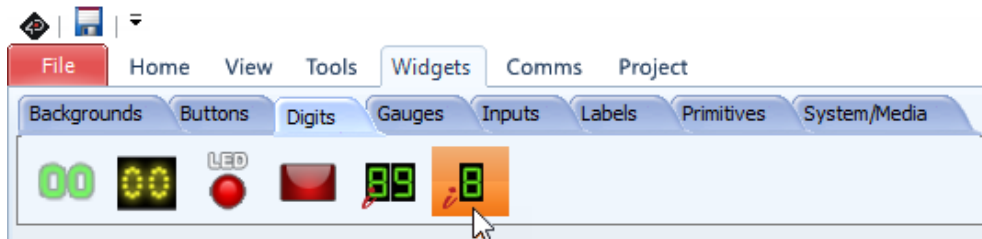
For instructions on how to create a new **ViSi** project, please refer to the section “**Create a New Project**” of any of the following application notes:

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

Design the Project

Add Internal LedDigit

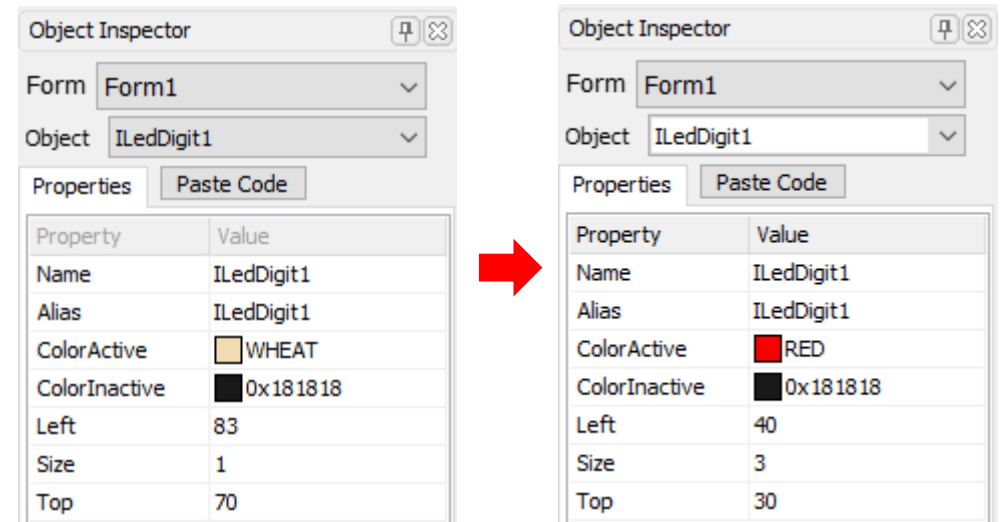
Add an Internal LedDigit to the Form by clicking on the Digits tab, then selecting the Internal LedDigits as shown below.



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

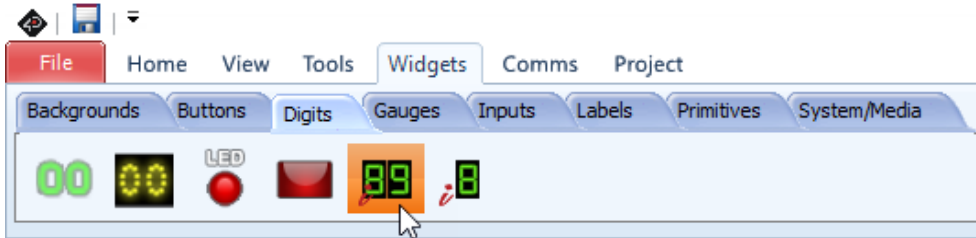


Go to the **Object Inspector** and modify the Internal LedDigit properties as needed.



Add Internal LedDigits

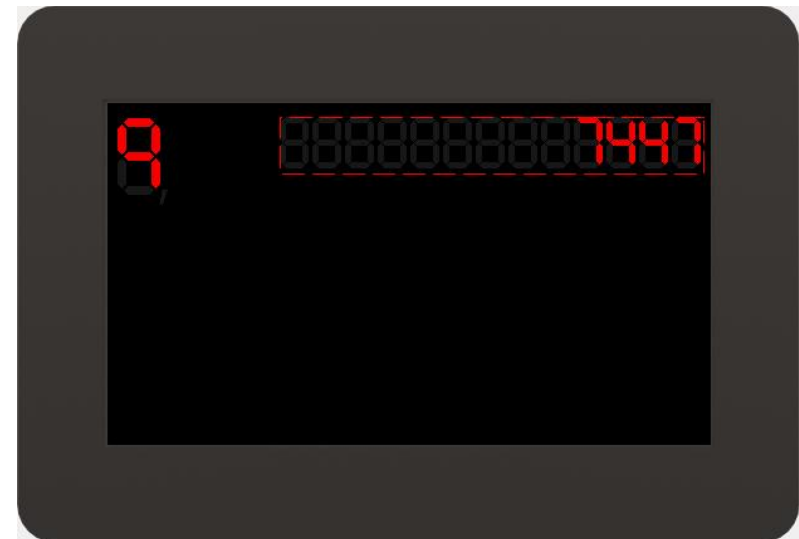
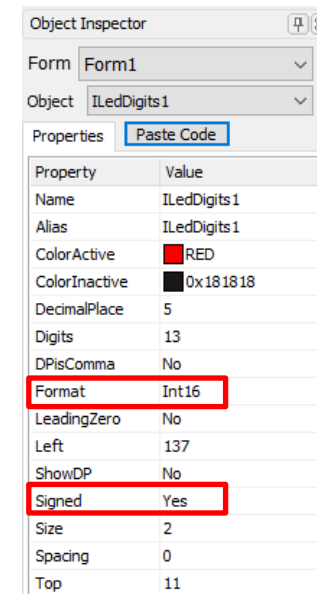
Add an Internal LedDigit to the Form by clicking on the Digits tab, then selecting the Internal LedDigits as shown below.





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



Go to the **Object Inspector** and modify the Internal LedDigit properties as shown. The first LedDigits is set with a signed 16-bit digit format (**Int16**).





Add a second LedDigits with a signed 32-bit digit format (**Int32**).

Property	Value
Name	ILedDigits2
Alias	ILedDigits2
ColorActive	 YELLOW
ColorInactive	 0x181818
DecimalPlace	5
Digits	13
DPisComma	No
Format	Int32
LeadingZero	No
Left	137
ShowDP	No
Signed	Yes
Size	2





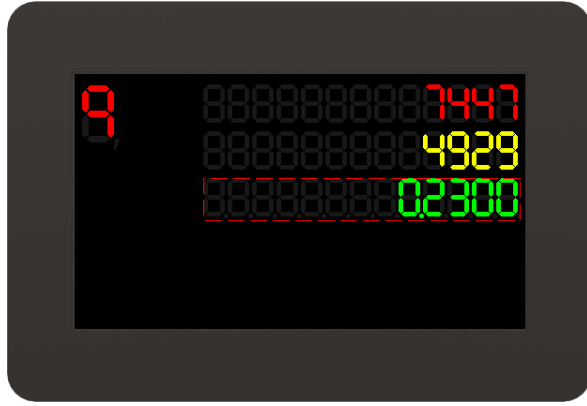
Add a fourth LedDigits with a signed general float format (**Float/Gen**).

Property	Value
Name	ILedDigits4
Alias	ILedDigits4
ColorActive	 AQUA
ColorInactive	 0x181818
DecimalPlace	5
Digits	13
DPisComma	No
Format	Float/Gen
LeadingZero	No
Left	137
ShowDP	Yes
Signed	Yes
Size	2





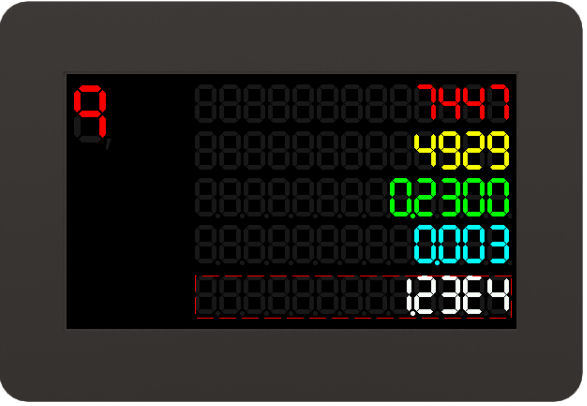
Add a third LedDigits with a signed decimal float format (**Float/Dec**).

Property	Value
Name	ILedDigits3
Alias	ILedDigits3
ColorActive	 LIME
ColorInactive	 0x181818
DecimalPlace	5
Digits	13
DPisComma	No
Format	Float/Dec
LeadingZero	No
Left	137
ShowDP	Yes
Signed	Yes
Size	2



Add a fifth LedDigits with a signed scientific float format (**Float/Sci**).

Property	Value
Name	ILedDigits5
Alias	ILedDigits5
ColorActive	 WHITE
ColorInactive	 0x181818
DecimalPlace	5
Digits	13
DPisComma	No
Format	Float/Sci
LeadingZero	No
Left	137
ShowDP	Yes
Signed	Yes
Size	2



Programming the Display

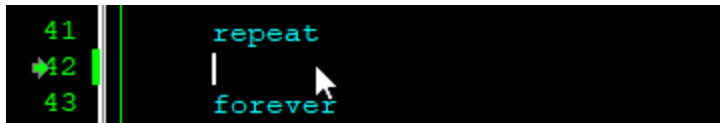
It is always ideal to prepare the graphics interface before moving to programming the display. This lessens the number of times the users will have to proceed with building and coping the graphics files to the uSD card.

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

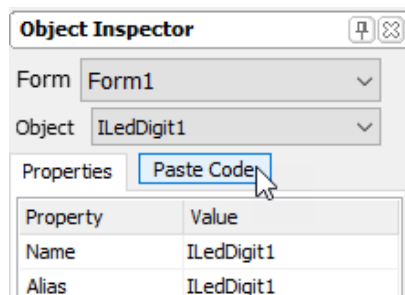
Paste Internal LedDigit Code

Similar to other widgets, ViSi provides Paste Code feature for Internal LedDigit and Internal LedDigits

Place the blinking cursor inside the repeat-forever loop.



Go to the Object Inspector, choose ILedDigit1 and click **Paste Code**



```
repeat
// ILedDigit1 1.0 generated 01/02/2020 1:00:00 PM
gfx_LedDigit(8, 12, 3, RED, 0x18C3, value) ;
```

Notice the fixed integers and colour included in the parameters of the ILedDigit1 function **gfx_LedDigit**. These values are based on computed values and may have to be regenerated by using the Paste Code again if the Internal LedDigit parameters is changed.

```
gfx_LedDigit(8, 12, 3, RED, 0x18C3, value) ;
```

The generated code includes the variable **value** which is the value that will update the widget display. This variable needs to be declared/changed to compile the project properly.

```
gfx_LedDigit(8, 12, 3, RED, 0x18C3, uint8) ; // 8-bit single
```

This value can be used directly as the hexadecimal digit or can be used to light up LED segment combinations. There is also an option for displaying the selected decimal point (dot or comma). These will be discussed in the following section.

Operate the LedDigit

Hexadecimal Digits

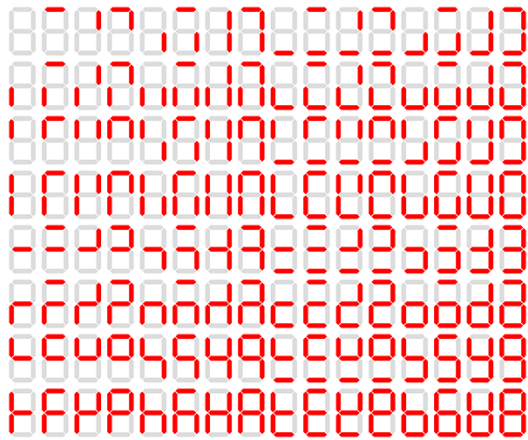
The LedDigit widget can be tested by updating the value parameter. The LedDigit widget is composed of one hexadecimal digit/character (0 to 15 or 0x00 to 0x0F). In this example, the digit is cycled from 0 to 15. Any values after 15 will light up random segments.

```
repeat
// Single LedDigit
uint8++; //Increment value for the single digit LED Digit
if (uint8 > 15) uint8 := 0; // Reset to 0 when maxed out

gfx_LedDigit(8, 12, 3, RED, 0x18C3, uint8) ; // 8-bit single
```

Individual Segments

The LedDigit widget also has a mode in which the value corresponds to the 128 possible LED segment combinations.



This can be done by adding the flag **LEDDIGIT_F_SET_SEGMENTS** to the input value.

```
// Single LedDigit
uint8++; //Increment value for the single digit LED Digit
if (uint8 > 128) uint8 := 0; // Reset to 0 when maxed out
txt_MoveCursor(0,0);
print(uint8);

gfx_LedDigit(8, 12, 3, RED, 0x18C3, uint8 + LEDDIGIT_F_SET_SEGMENTS) ;
```

Decimal Point

The LedDigit widget can also have its decimal point enabled by adding the **LEDDIGIT_F_SHOW_DP** flag to the input value.



```
gfx_LedDigit(8, 12, 3, RED, 0x18C3, uint8 + LEDDIGIT_F_SHOW_DP) ;
```

The decimal point can also be changed into a comma by adding the **LEDDIGIT_F_DP_COMMA** flag to the input value.



```
uint8 + LEDDIGIT_F_SHOW_DP + LEDDIGIT_F_DP_COMMA) ;
```

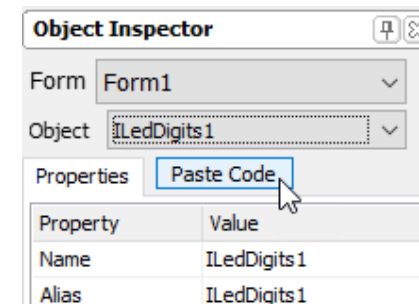
The decimal point can be activated up by adding the **LEDDIGIT_F_DP_ON** flag to the input value.



```
uint8 + LEDDIGIT_F_SHOW_DP + LEDDIGIT_F_DP_ON) ;
```

Paste Internal LedDigits Codes

Go to the Object Inspector, choose ILedDigits1 and click **Paste Code**




```
#inherit "gfx_LedDigits.inc"

#DATA
    // ILedDigits1 Data Start
    word IILedDigits1 40, 160, 156, 69, 4, -1, 0, 3, RED, 0x18C3, 0x0
    // ILedDigits1 Data End
#END
var IILedDigits1RAM[WIDGET_RAM_SPACE+2] ; // ledDigits has 2 values and
```

The inserted code automatically inserts the LedDigits parameters in a Data Block format and the required RAM allocation at the start of the project.

```
gfx_LedDigit(10, 50, 0, RED, 0x18C3, value, );

// ILedDigits1 1.0 generated 01/02/2020 11:00:05 AM
gfx_LedDigits(posn, IILedDigits1RAM, IILedDigits1) ;
forever
endfunc
```

Notice the fixed integers and colour are included in the data block for the IILedDigits1 function **gfx_LedDigit**. These values are based on computed values and may have to be regenerated by using the Paste Code button again if the Internal LedDigits parameters is changed.

Paste the code of the remaining four LedDigits in the Form.

```
gfx_LedDigits(posn, IILedDigits1RAM, IILedDigits1) ;

gfx_LedDigits(posn, IILedDigits2RAM, IILedDigits2) ;

gfx_LedDigits(posn, IILedDigits3RAM, IILedDigits3) ;

gfx_LedDigits(posn, IILedDigits4RAM, IILedDigits4) ;

gfx_LedDigits(posn, IILedDigits5RAM, IILedDigits5) ;
forever
endfunc
```

Displaying the LedDigits Value

The LedDigits widget can accept signed or unsigned 16-bit, 32-bit or Float values depending on the format selected.

In this example we will generate a random 32-bit signed value

```
// Generate Random unsigned Long value
uvar32[0] := RAND();
uvar32[1] := RAND();
```

and use Float constants to update each widget with different format

```
// Float Values
flt_VAL(f1,"3.141592653589");
flt_VAL(f2,"6.0221409e5");
flt_VAL(f3,"1.602176634e-5");
flt_VAL(f4,"-9000.34567");
flt_VAL(f5,"0.05");

floats[0] := f1;
floats[1] := f2;
floats[2] := f3;
floats[3] := f4;
floats[4] := f5;
```

The first LedDigits is set with a signed 16-bit digit format (**Int16**). The low word of the 32-bit long value is used as the 16-bit input for this LedDigits.

```
gfx_LedDigits(uvar32[0], IILedDigits1RAM, IILedDigits1) ; // 16-bit format (signed)
```

The second LedDigits is set with a signed 32-bit digit format (**Int32**). The variable array containing the 32-bit value can be used directly in the LedDigits function.

```
gfx_LedDigits(uvar32, IILedDigits2RAM, IILedDigits2) ; // 32-bit format (signed)
```

The three remaining LedDigits with Float format can directly receive float formatted values that can be generated manually or through the use of 4DGL float functions. In this application note, the float constants to be displayed are just cycled through.

```
// Cycle through each Float value
i++;
if (i > 4) i := 0;
gfx_LedDigits(floats[i], ILedDigits3RAM, IILedDigits3) ; // Float with fixed decimal place

gfx_LedDigits(floats[i], ILedDigits4RAM, IILedDigits4) ; // Float with limited significant

gfx_LedDigits(floats[i], ILedDigits5RAM, IILedDigits5) ; // Float with scientific notation
```

Pause the Loop

It is sometimes necessary to add delays to certain part of projects in order to observe changing values. Add a small delay inside the loop to perform an update only every 400ms by using the **pause()** function.

```
    pause(400); // Delay 400 ms
  forever
endfunc
```

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

Demonstration

After uploading the code to the display module, the LED digits should display on the screen as shown below.



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.