



ViSi The LED Digits Object

DOCUMENT DATE: **15th April 2019**
DOCUMENT REVISION: **1.1**



Description

This application note is intended to demonstrating to the user the interconnection of the 4D Systems Diablo16 display module with a ZIGBEE personal area network module.

Before getting started, the following are required:

- Any of the following 4D Picaso display modules:

uLCD-24PTU	uLCD-28PTU	uVGA-III
gen4-uLCD-24PT	gen4-uLCD-28PT	gen4-uLCD-32PT

and other superseded modules which support the Designer and/or ViSi environments.

- The target module can also be a Diablo16 display

gen4-uLCD-24D	gen4-uLCD-28D	gen4-uLCD-32D
Series	Series	Series
gen4-uLCD-35D	gen4-uLCD-43D	gen4-uLCD-50D
Series	Series	Series
gen4-uLCD-70D		
Series		
uLCD-35DT	uLCD-43D Series	uLCD-70DT

Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.

- [4D Programming Cable](#) / [μUSB-PA5/μUSB-PA5-II](#) for non-gen4 displays (uLCD-xxx)
- [4D Programming Cable](#) & [gen4-IB](#) / [gen4-PA](#) / [4D-UPA](#), for gen-4 displays (gen4-uLCD-xxx)
- [micro-SD \(μSD\)](#) memory card
- [Workshop 4 IDE](#) (installed according to the installation document)
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

Content

Description	2	Float Variables are not Allowed	12
Content	3	<i>Modify the Properties of the LED Digits Object</i>	12
Application Overview	4	Change the Decimal Precision	12
Setup Procedure	4	Change the Number of Digits	13
Create a New Project	4	Resize and Move to a New Location	13
Design the Project	5	Change the Colour	13
<i>Add a LED Digits Object to Form1</i>	5	Leading Zeros	14
<i>Uncomment the uSD Card Initialization Routine</i>	5	<i>Regenerate and Repaste the Code</i>	14
<i>Paste and Modify the Code</i>	6	<i>Update the Contents of the uSD Card</i>	14
<i>The Include File</i>	7	<i>Run basicLED2</i>	14
The Definition for ledDigitsDisplay(...)	8	<i>Limitations of the LED Digits Object</i>	15
<i>Initialize the Variable "numx"</i>	9	Displaying Values Higher than 32,767	15
<i>Insert a Clear Screen Command</i>	9	Displaying Hexadecimal Numbers	15
<i>Run basicLED</i>	9	Displaying of Numbers with a Base Lower than 10	15
<i>Use a Loop to Drive the LED Digits Object</i>	10	<i>Displaying Negative Values</i>	15
<i>Frames of Objects</i>	10	Change the Include File	16
Frames of a LED Digits Object	10	Specify the Correct Number of Digits	16
<i>Display a LED Digits Object Frame</i>	11	Displaying Values Lower than -32,768	16
Display the First Frame	11	Run the Program	16
Display the nth Frame	11	Proprietary Information	17
Frame Index vs. Value Displayed	12	Disclaimer of Warranties & Limitation of Liability	17

Application Overview

The LED digits object or widget is very useful in displaying numerical values. The digits in a LED digits object look like those in an actual seven segment display. The user can modify the width, height, number of digits, decimal precision, colour, and position of a LED digits object. The appearance however is essentially fixed and cannot be modified. Users wanting to display digits with a customised appearance will have to use the custom digits object. The custom digits object requires the user to specify a bitmap image from which the digits will be taken. The bitmap image is an image strip containing the digits 0 to 9. To learn more about the custom digits object, refer to the application note [ViSi The Custom Digits Object](#).

This application note discusses the basic properties of the LED digits object and shows how to drive the object using 4DGL routines. The limitations of the LED digits object are then discussed together with their suggested solutions. This application notes comes with three demo projects.

Name	Description
basicLED	Shows the basics of using the LED digits object
basicLED2	Shows the basics of using the LED digits object
basicLED3	Shows how the LED digits object is used to display negative values.

Setup Procedure

For instructions on how to launch Workshop 4, 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](#)

Create a New Project

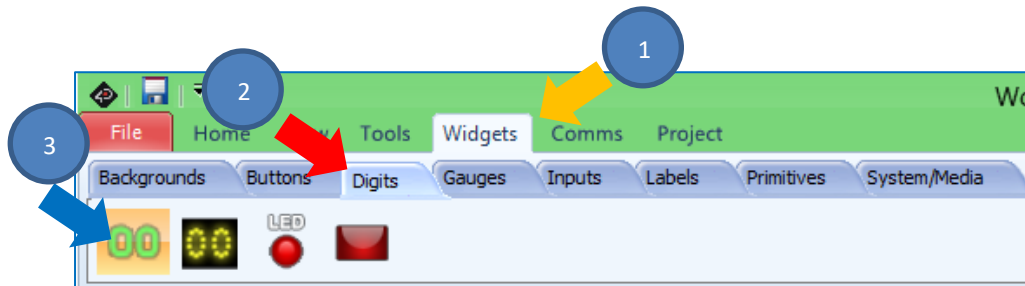
For instructions on how to create a new **ViSi** project, please refer to the section “**Create a New Project**” of the application note

[ViSi Getting Started - First Project for Picaso and Diablo16](#)

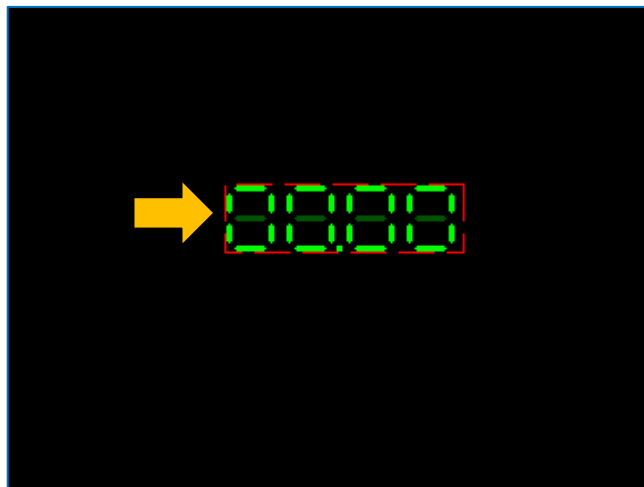
Design the Project

Add a LED Digits Object to Form1

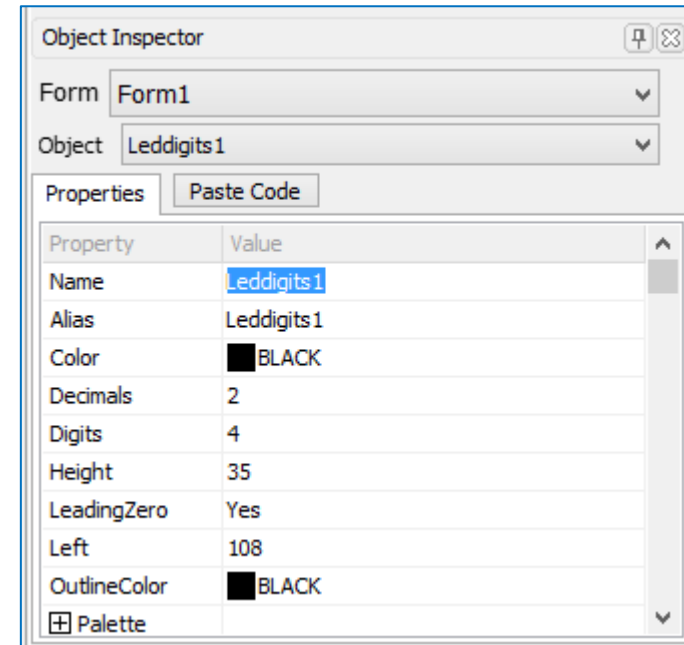
Select the LED digits icon as shown below.



Click on the WYSIWYG screen to place the object.

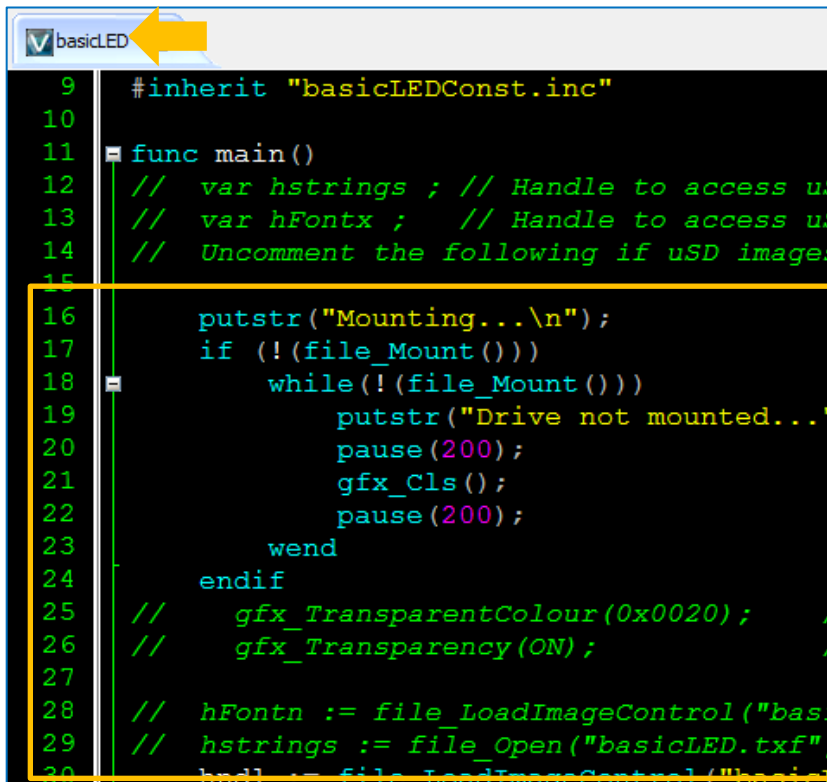


The Object Inspector shows the property of the newly created LED digits object – **Leddigits1**.



Uncomment the uSD Card Initialization Routine

Note that the uSD card initialization routine has been uncommented and that the project has been saved with the name "basicLED". If not familiar with how to do these, refer to the application note [ViSi Getting Started - First Project for Picaso and Diablo16](#).



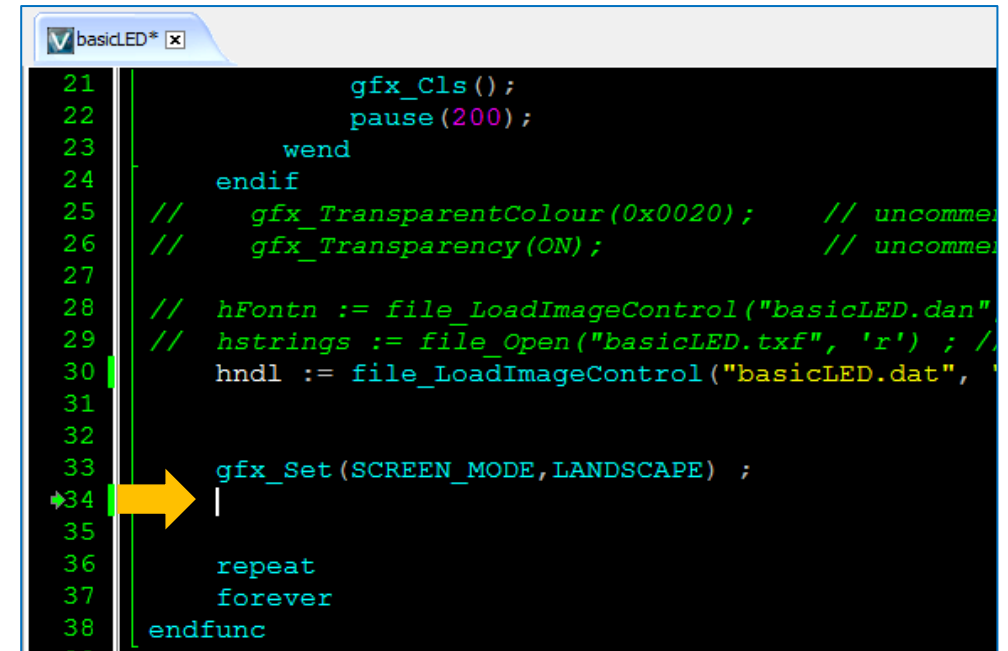
```

9  #inherit "basicLEDConst.inc"
10
11 func main()
12 // var hstrings ; // Handle to access u
13 // var hFontx ; // Handle to access u
14 // Uncomment the following if uSD image.
15
16 putstr("Mounting...\n");
17 if (!(file_Mount()))
18 while (!(file_Mount()))
19     putstr("Drive not mounted...");
20     pause(200);
21     gfx_Cls();
22     pause(200);
23 wend
24 endif
25 // gfx_TransparentColour(0x0020);
26 // gfx_Transparency(ON);
27
28 // hFontn := file_LoadImageControl("bas
29 // hstrings := file_Open("basicLED.txf"
30 hndl := file_LoadImageControl("basic

```

Paste and Modify the Code

With the uSD card routine uncommented in the default code, place the cursor just after the screen orientation initialization line like as shown below.

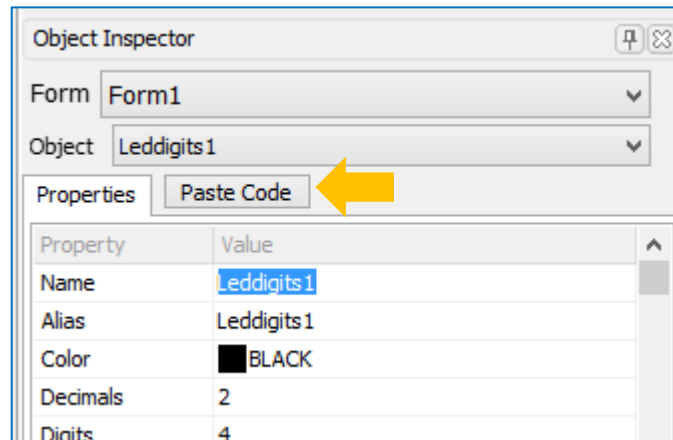


```

21     gfx_Cls();
22     pause(200);
23 wend
24 endif
25 // gfx_TransparentColour(0x0020); // uncommen
26 // gfx_Transparency(ON); // uncommen
27
28 // hFontn := file_LoadImageControl("basicLED.dan"
29 // hstrings := file_Open("basicLED.txf", 'r') ; //
30 hndl := file_LoadImageControl("basicLED.dat", '
31
32
33     gfx_Set(SCREEN_MODE, LANDSCAPE) ;
34
35
36     repeat
37     forever
38 endfunc

```

Click on the Paste Code button in the Object Inspector.



The code area should be updated accordingly.

```
gfx_Set (SCREEN_MODE, LANDSCAPE) ;
// Leddigits1 1.0 generated 8/29/2015 7:54:38 AM
img_Show(hndl, iLeddigits1); // show all digits at 0, on
ledDigitsDisplay(numx, iLeddigits1+1, 108, 4, 3, 30, 0)
```

The first line – **img_Show(...)** – simply displays the initial frame of Leddigits1 which is its “zero” state. The second line – **ledDigitsDisplay(...)** – displays a frame of Leddigits1. The variable “**numx**” determines the frame to be displayed.

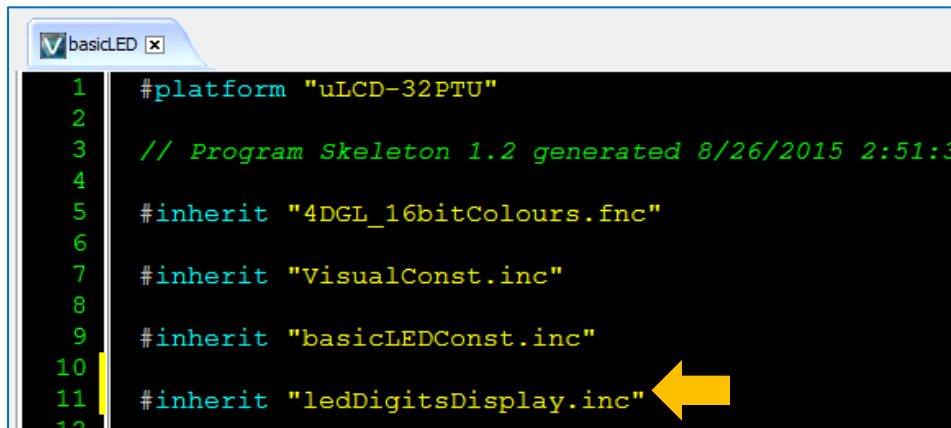
Note: The remaining paramters for ledDigitsDisplay(...) are automatically copied from the Object Inspector. This means that if we change any property of a LED digits object either thru the WYSIWYG screen or the Object Inspector, we would need to update its ledDigitsDisplay(...) line and all other instances of this line in the code. We would also need to update the contents of the uSD card. Otherwise, the LED digits object will be rendered incorrectly.

The Include File

The function “**ledDigitsDisplay(...)**” is not internal to either the Picaso or Diablo16 processor. It is defined in a separate header or include file. Compiling the code at this point would yield the error shown below.

```
Error: 'ledDigitsDisplay' not found (line 36
file:basicLED.4Dg)
```

To correct this, include the header file “`ledDigitsDisplay.inc`” in the code, like as shown below.



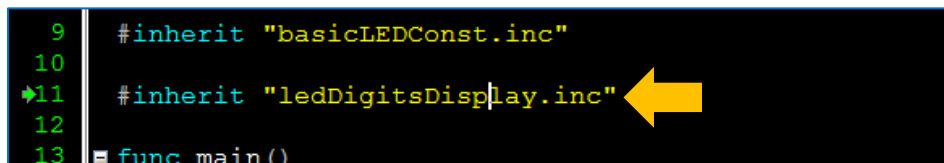
```

1 #platform "uLCD-32PTU"
2
3 // Program Skeleton 1.2 generated 8/26/2015 2:51:3
4
5 #inherit "4DGL_16bitColours.fnc"
6
7 #inherit "VisualConst.inc"
8
9 #inherit "basicLEDConst.inc"
10
11 #inherit "ledDigitsDisplay.inc"
12

```

The Definition for `ledDigitsDisplay(...)`

Put the cursor on the filename text of `ledDigitsDisplay.inc`.

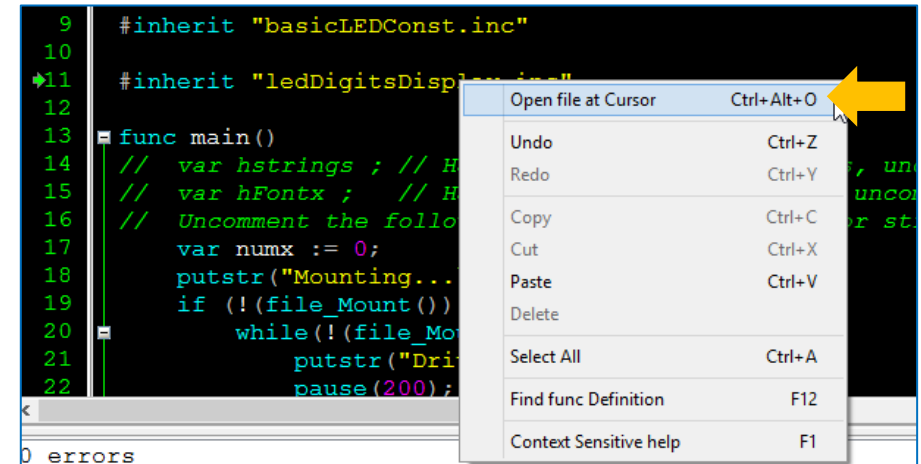


```

9 #inherit "basicLEDConst.inc"
10
11 #inherit "ledDigitsDisplay.inc"
12
13 func main()

```

Click on the right mouse button and choose the first option – **Open file at Cursor**.

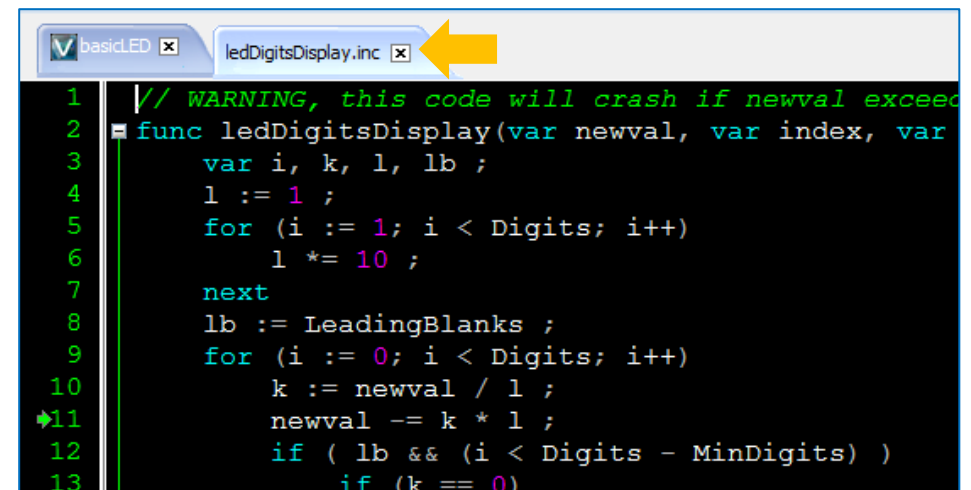


```

9 #inherit "basicLEDConst.inc"
10
11 #inherit "ledDigitsDisplay.inc"
12
13 func main()
14 // var hstrings ; // H
15 // var hFontx ; // H
16 // Uncomment the follo
17 var numx := 0 ;
18 putstr("Mounting...
19 if (!(file_Mount())
20 while (!(file Mo
21 putstr("Dri
22 pause(200);

```

The file `ledDigitsDisplay.inc` should now open in another tab. Inside the file is the definition for the function `ledDigitsDisplay(...)`.



```

1 // WARNING, this code will crash if newval exceed
2 func ledDigitsDisplay(var newval, var index, var
3 var i, k, l, lb ;
4 l := 1 ;
5 for (i := 1; i < Digits; i++)
6 l *= 10 ;
7 next
8 lb := LeadingBlanks ;
9 for (i := 0; i < Digits; i++)
10 k := newval / l ;
11 newval -= k * l ;
12 if ( lb && (i < Digits - MinDigits) )
13 if (k == 0)

```


Understanding the definition of the function `ledDigitsDisplay(...)`, although not really necessary at this point, is left to the reader as an exercise.

Initialize the Variable “numx”

Workshop automatically includes the variable “numx” when it generates the code for a LED digits object.

```
// Leddigits1 1.0 generated 8/29/2015 6:58:29 AM
img_Show(hndl, iLeddigits1); // show all digits at 0, 0
ledDigitsDisplay(numx, iLeddigits1+1, 108, 4, 3, 30, 0)
```

Now this is not yet defined in the code, so we will have to declare and initialize it before using it, like as shown below.

```
13 func main()
14 // var hstrings ; // Handle to access uSD string
15 // var hFontx ; // Handle to access uSD fonts,
16 // Uncomment the following if uSD images, fonts
17 var numx := 0;
18 putstr("Mounting...\n");
19 if (hFile_Mount())
```

Alternatively, we can also write:

```
ledDigitsDisplay(0, iLeddigits1+1, 108, 4, 3, 30, 0) ;
```

Insert a Clear Screen Command

Insert a clear screen command between the screen orientation initialization line and the code block for `Leddigits1`. This is to clear the strings printed during the uSD card initialization routine.

```
34
35 gfx_Set (SCREEN_MODE, LANDSCAPE) ;
36 gfx_Cls () ;
37
```

Run basicLED

We are now ready to run the project. If not familiar with how to run a ViSi project, proceed to the section “Run the Program”. The display module should now render the first frame of `Leddigits1`, which, in this case, happens to correspond to the value “00.00”.



Use a Loop to Drive the LED Digits Object

Below, a repeat-until loop is used to make Leddigits1 display the values 0 until 99.

```
// Leddigits1 1.0 generated 8/29/2015 7:54:38 AM
img_Show(hndl, iLeddigits1); // show all digits at 0, only d
ledDigitsDisplay(numx, iLeddigits1+1, 108, 4, 3, 30, 0) ;

repeat
  numx++;
  ledDigitsDisplay(numx, iLeddigits1+1, 108, 4, 3, 30, 0) ;
  pause(100);
until(numx == 100);

repeat
forever
```

Note that we simply copied the generated ledDigitsDisplay(...) line for Leddigits1 and pasted it inside a repeat-until loop. Recompile the code and download it again to the display module's processor. Leddigits1 should render its frames 0 to 99, which correspond to the values 00.00 to 00.99, respectively.

Note also that only the code was modified (and not Leddigits1 in the WYSIWYG area nor any of its properties in the Object Inspector). If only the code alone is modified and not the object itself, there is no need to update the contents of the uSD card mounted on the display module.

Frames of Objects

Objects or widgets can be classified as single-frame or multi-frame. Examples of single-frame objects are those under the Backgrounds pane (border, gradient, and scale), the label, the static text, and the image object. These objects have only one frame of image inside them. The rest of the objects are multi-frame since they contain two or more images. For example, most of the objects under the Buttons pane are two-frame objects. A meter object with minimum and maximum values of 0 and 100, respectively, would have 101 frames of images inside it – one frame of image to represent each state of the meter.

Frames of a LED Digits Object

Similarly, we can think of the LED digits object as containing multiple frames, the number of which depends on the number of digits. A four-digit LED digits object for example would have 10,000 frames of images inside it, since it should be able to display all values between 0 and 9999.

In reality, the implementation of a LED digits object is quite different compared to other objects. For instance, the meter object example above has exactly 101 frames of images inside it. On the other hand, a LED digits object, of any number of digits, has actually only 12 frames of images inside it – one for each digit (0 to 9), one for a “blank digit”, and one for the negative sign. Added to these, another image is created to represent the “container” frame. The function ledDigitsDisplay(...) is used to properly render the digits of a LED digits object to display any given value. To learn more about the details of how a LED digits object is implemented, interested readers may study the definition for the function ledDigitsDisplay(...), which

can be found by following the instructions described in the previous section “**The Include File**”.

To facilitate this discussion, however, it is convenient at this point for the reader to think that a four-digit LED digits object has 10,000 frames of images inside of it, that these frames are indexed from 0 to 9,999, and that each frame can be displayed by specifying its index value. We then use the function `ledDigitsDisplay(...)` to display the desired frame, the index of which is determined by the first parameter passed to the said function.

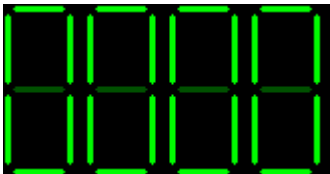
Display a LED Digits Object Frame

Display the First Frame

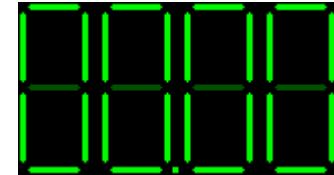
To display the first frame of a LED digits object, we write

```
ledDigitsDisplay(0, iLeddigits1+1, 108, 4, 3, 30, 0);
```

A four-digit LED digits object with no decimal digits would look like as that shown in the image below.



A four-digit LED digits object with two decimal digits would look like as that shown in the image below.

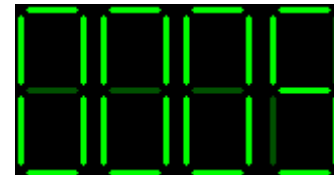


Display the nth Frame

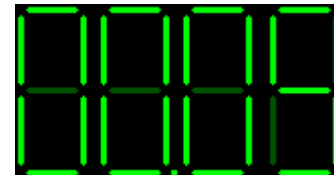
To display the sixth frame, for example, of a LED digits object, we write

```
ledDigitsDisplay(5, iLeddigits1+1, 108, 4, 3, 30, 0);
```

A four-digit LED digits object with no decimal digits would look like as that shown in the image below.



A four-digit LED digits object with two decimal digits would look like as that shown in the image below.



Frame Index vs. Value Displayed

Note that the first parameter for `ledDigitsDisplay(...)` corresponds to the frame index and not necessarily to the value being displayed. The first frame may correspond to the value "0.0" or "0" depending on the properties of the LED digits object. Likewise, the sixth frame, whose index value is 5, may correspond to either "0.05" or "5", which are not equivalent.

Float Variables are not Allowed

Also, as of writing, only an integer variable can be used as the first parameter for the function `ledDigitsDisplay(...)`. Thus, we cannot use a float variable or an array as the first parameter for `ledDigitsDisplay(...)`. To be able to display decimal values, set the decimal precision property of the LED digits object accordingly. Simple arithmetic may then be needed in the code so that decimal values can be rendered using integers to specify the correct frame index.

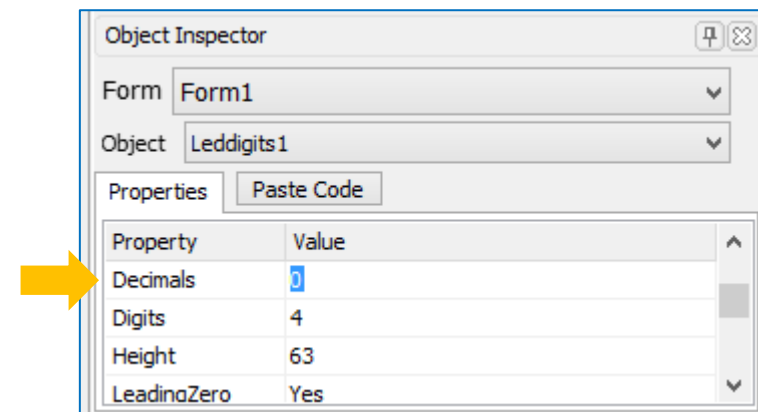
Name	Leddigits1
Alias	Leddigits1
Color	BLACK
Decimals	2
Digits	4

Modify the Properties of the LED Digits Object

We will now modify the properties of `Leddigits1`. Note again that any modification done on `Leddigits1` will require the uSD card contents and the lines related to `Leddigits1` in the code area to be updated.

Change the Decimal Precision

The decimal precision is changed from 2 to 0.

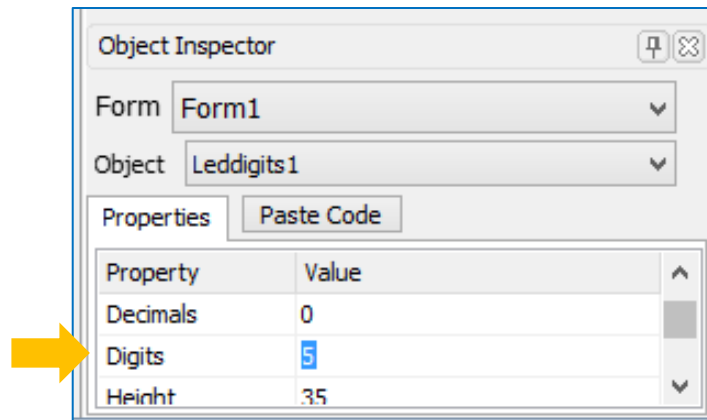


The WYSIWYG area should be updated accordingly.



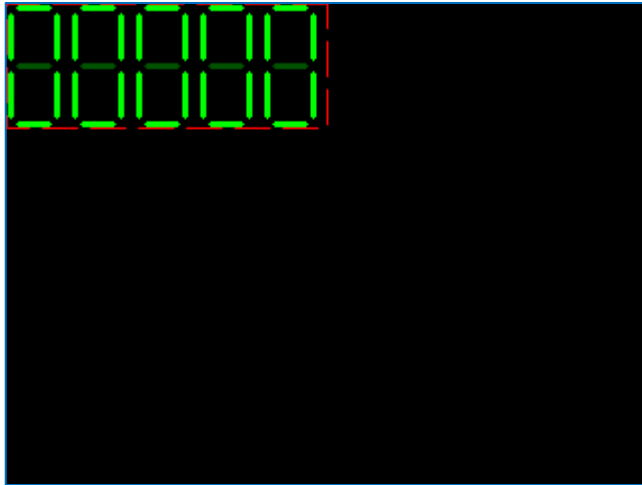
Change the Number of Digits

The value of the Digits property is changed from 4 to 5.



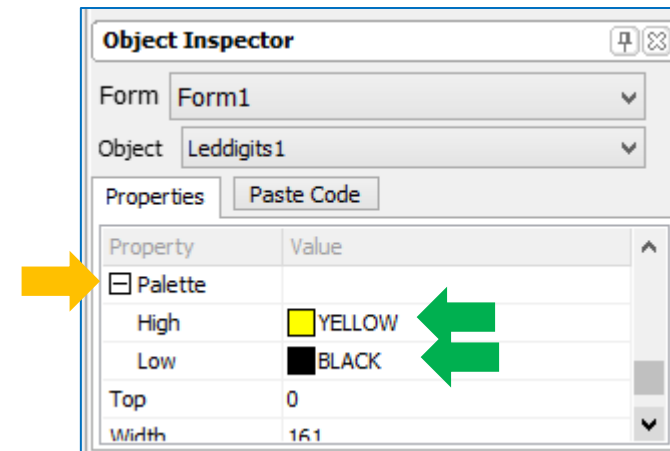
Resize and Move to a New Location

The edges of the box around Leddigits1 can be dragged to resize the object. Also, the object itself can be dragged to another location in the WYSIWYG area. The result is shown below.

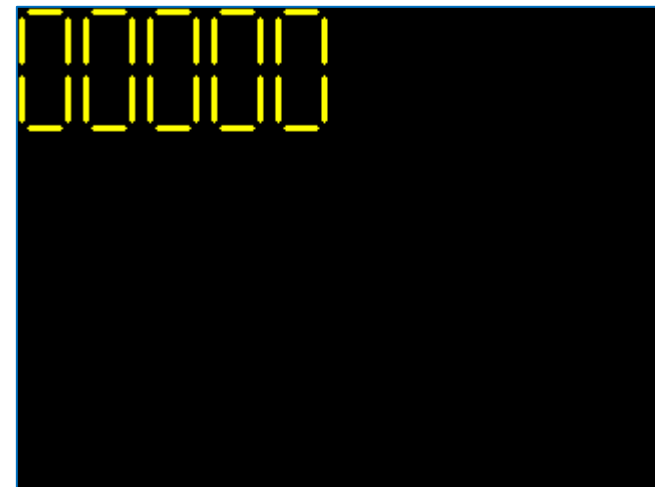


Change the Colour

Change the colour like as shown below.

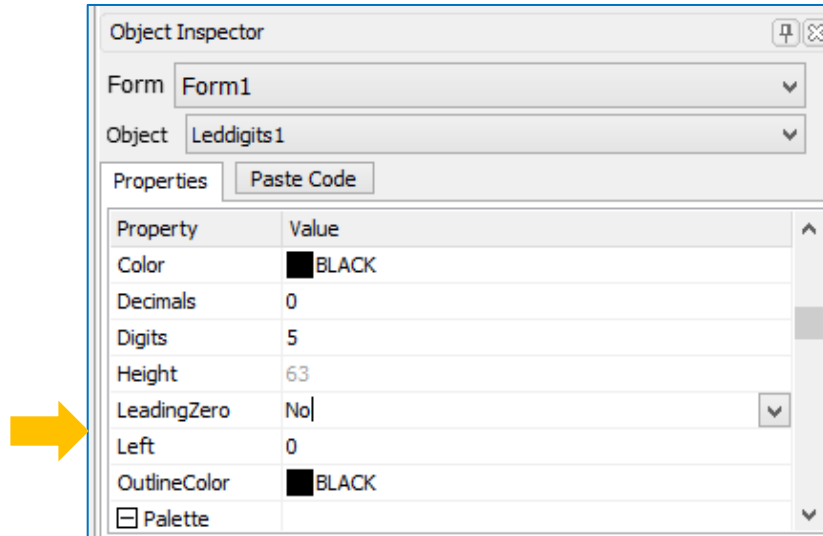


The WYSIWYG area is updated accordingly.

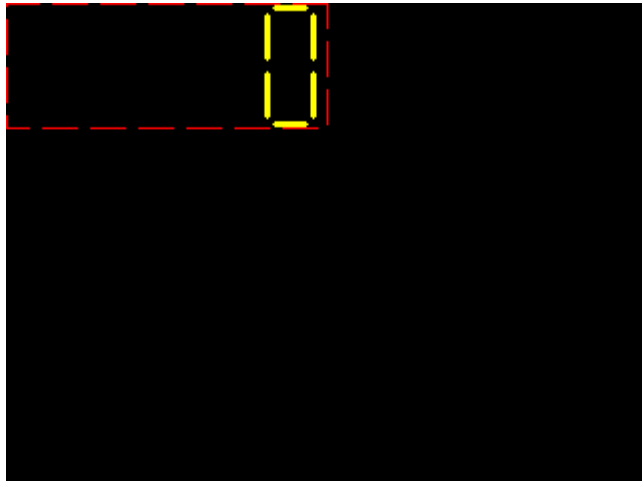


Leading Zeros

The LeadingZero property determines whether or not the insignificant leading zeros will be displayed.



The WYSIWYG area is updated accordingly.



Regenerate and Repaste the Code

In the image below, the old ledDigitsDisplay(...) lines are commented out and replaced with the updated version. The updated version can be generated by clicking again on the Paste Code button of the Object Inspector.

The screenshot shows the code in the Object Inspector. The old code is commented out, and the new code is pasted. A yellow arrow points to the new code, and a green arrow points to the old code.

```
// Leddigits1 1.0 generated 8/29/2015 7:54:38 AM
img_Show(hndl, iLeddigits1); // show all digits at 0, only
//ledDigitsDisplay(numx, iLeddigits1+1, 108, 4, 3, 30, 0) ;
ledDigitsDisplay(numx, iLeddigits1+1, 0, 5, 1, 32, 1) ;

repeat
    numx++;
//ledDigitsDisplay(5, iLeddigits1+1, 108, 4, 3, 30, 0);
ledDigitsDisplay(numx, iLeddigits1+1, 0, 5, 1, 32, 1) ;
    pause(100);
until(numx == 100);
```

Update the Contents of the uSD Card

Any change made on the WYSIWYG area or the Object Inspector will cause Workshop to generate a new set of supporting files for the project. Thus, we will need to unmount the uSD card from the display module, mount it to the PC, and let Workshop update the files.

Run basicLED2

Finally we compile the code and download the program to the processor. With the updated uSD card mounted back to the display module, the new program should now run. When the program runs, Leddigits1 should be larger and should be on the top-left corner of the display. It should be yellow in colour and the insignificant leading zeros should not be displayed. Attached is another project – **basicLED2**. The project **basicLED2** has the

yellow LED digits object in it while **basicLED** has the green LED digits object from our first example.

Limitations of the LED Digits Object

It would be logical to say that a five-digit LED digits object could display all values between 0 and 99,999 – a total of 100,000 frames. Remember however that the Picaso and Diablo16 are 16-bit processors, which means that the “normal” width of an integer data type is 16 bits. Signed number operations further limit the maximum positive value of an integer variable to 32,767 ($[2^{16}/2]-1$), since the other half of the range is used to represent negative integer values. The maximum positive value therefore that an integer variable can store is 32,767. This is also the maximum number that the function “**ledDigitsDisplay(...)**”, defined in the include file “**ledDigitsDisplay.inc**”, can correctly render. The minimum value that can be rendered by this function is zero. Using this function, attempting to write a value beyond than these limits to a LED digits object would result to frames with red X marks appearing on the object. The red X marks indicate that the frame being accessed does not exist. Also, it wouldn’t make sense to create a LED digits object with six or more digits in it.

Displaying Values Higher than 32,767

However, since the implementation of a LED digits object is different as described in the section “**Frames of a LED Digits Object**”, it is possible to create a routine that can render 32-bit values. For more information, refer to the application note [ViSi Displaying Large Integers with the LED Digits Object](#).

Displaying Hexadecimal Numbers

As of writing, the LED digits object can render decimal values (base 10) only. Displaying of hexadecimal values is not yet supported.

Displaying of Numbers with a Base Lower than 10

Displaying of numbers whose base is lower than 10, such as octal and binary numbers, would be possible by using a correctly modified version of the `ledDigitsDisplay(...)` function.

Displaying Negative Values

As previously mentioned, the `ledDigitsDisplay(...)` function defined in the include file “`ledDigitsDisplay.inc`” can render values between 0 and 32,767 (inclusive). It cannot, however, render values between -1 and -32,768 (inclusive). To be able to render negative values, we need to use the `ledDigitsDisplay(...)` function defined in **another** include file. This function can correctly render values between -32,768 and 32,767 (inclusive).

Change the Include File

Attached is the project “**basicLED3**”. This project shows how a LED digits object is used to display negative numbers. First, note that we are now using a different include file.

```
5  #inherit "4DGL_16bitColours.fnc"
6
7  #inherit "VisualConst.inc"
8
9  #inherit "basicLED3Const.inc"
10
11 #inherit "ledDigitsDisplay-ve.inc" ←
12
```

Specify the Correct Number of Digits

Second, note that the number of digits of Leddigits1 was changed from 5 to 6. This is because we want Leddigits1 to be able to display the value “**-32,768**”. Essentially, the negative sign requires an additional digit place. If the number of digits were still “5” and we attempted to display the value “**-32,768**”, the negative sign would be superimposed on the most significant digit, like as shown below.



Therefore, when using the `ledDigitsDisplay(...)` function defined in the include file “`ledDigitsDisplay-ve.inc`” to display negative values, determine

first the minimum value that needs to be displayed and make sure that the correct number of digits is specified accordingly.

Displaying Values Lower than -32,768

The application note [ViSi Displaying Large Integers with the LED Digits Object](#) shows how a project that can display 32-bit values (signed and/or unsigned) is implemented.

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, 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](#)

The uLCD-32PTU and uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

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.