APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi 4DGL Strings Print Formats – the Long Decimal Format Specifiers

DOCUMENT DATE: **22nd April 2019**
DOCUMENT REVISION: **1.1**

Description

There are four decimal format specifiers:

Specifier	Data to be displayed
%d	Signed decimal
%u	Unsigned decimal
%ld	Long signed decimal
%lu	Long unsigned decimal

This application note discusses how the **long signed decimal** and **long unsigned decimal** format specifiers are used with the **str_Printf(…)** function. This application note is intended for use in the Workshop 4 – Designer environment. The 4DGL code of the Designer project can be copied and pasted to an empty ViSi project and it will compile normally. The code can also be integrated to that of an existing ViSi project.

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  and other superseded modules which support the Designer and/ or ViSi environments.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 Series	gen4-uLCD-28D Series	gen4-uLCD-32D Series
gen4-uLCD-35D Series	gen4-uLCD-43D Series	gen4-uLCD-50D 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

Application Overview

The application note Designer or ViSi Strings and Character Arrays explains how 4DGL strings and character arrays are stored in and accessed from memory. It also differentiates between word-aligned and byte-aligned pointers. Furthermore, it introduces the use of the function ***str_Printf(…)***.

The application note Designer or ViSi 4DGL Strings Print Formats – the String and Character Format Specifiers shows how the string and character format specifiers ("**%s**" and "**%c**", respectively) are used. Also, it covers the topics "**Automatic Advancing of the Pointer**" and "**Dynamic Construction of the Format Specifier**".

This application note now further explains the use of the ***str_Printf(…)*** function together with the long signed decimal and long unsigned decimal format specifiers.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **Designer** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note
Designer Getting Started - First Project

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 **Designer** project, please refer to the section "**Create a New Project**" of the application note
Designer Getting Started - First 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

The Format Specifier "%ld"

The format specifier "**%ld**" is used for displaying long signed decimal numbers. A long signed decimal in 4DGL is a signed 32-bit (or 4-byte) integer value, the range of which is from ***-2,147,483,648*** to ***2,147,483,647*** ***(*** -2^{31} to $2^{31}-1$). Consider the code snippet shown below.

```
var val32[2];
var ptr;

gfx_ScreenMode(LANDSCAPE) ;          // change manual

umul_1616(val32, 500, 2000);
ptr := str_Ptr(val32);

print("ptr old: ",ptr,"\n");

print("val32: ");
str_Printf(&ptr, "%ld");

print("\n");
print("ptr new: ",ptr);
```

The output of the above code is:

```
ptr old: 16
val32: 1000000
ptr new: 20
```

The function ***umul_1616(…)*** performs an unsigned multiplication of two 16-bit values, placing the 32-bit result in a two-word array. In this example, the two 16-bit values are 500 and 2000. When multiplied together the product of these is 1000000. If we print the contents of the word array ***val32*** in hexadecimal format,

```
print("\n");
print("val32[0]: ",[HEX]val32[0]);

print("\n");
print("val32[1]: ",[HEX]val32[1]);
```

we get,

```
ptr old: 16
val32: 1000000
ptr new: 20
val32[0]: 4240
val32[1]: 000F
```

We analyse the contents of the word array ***val32***.

element	val32[0]		val32[1]	
byte	*high*	*low*	*high*	*low*
address	*17*	*16*	*19*	*18*
Hex	**42**	**40**	**00**	**0F**

Low word High word

0x000F 4240 = 1000000

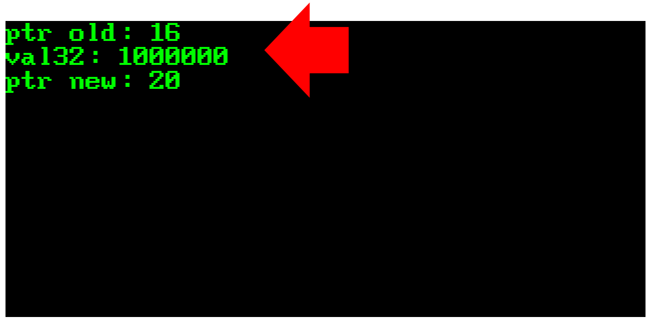
Note also that the pointer was advanced by four bytes after the long signed decimal value was printed.

element	val32[0]		val32[1]			
byte	*high*	*low*	*high*	*low*	*high*	*low*
Hex	**42**	**40**	**00**	**0F**	**-**	**-**
address	*17*	*16*	*19*	*18*	*21*	*20*

ptr old **ptr new**

Therefore, the long signed decimal format specifier, "**%ld**", causes the **str_Printf(…)** function to get four bytes from the address starting at that pointed to by the byte-aligned pointer. ***str_Printf(…)*** then treats these four bytes as a 32-bit signed integer and prints the decimal equivalent.

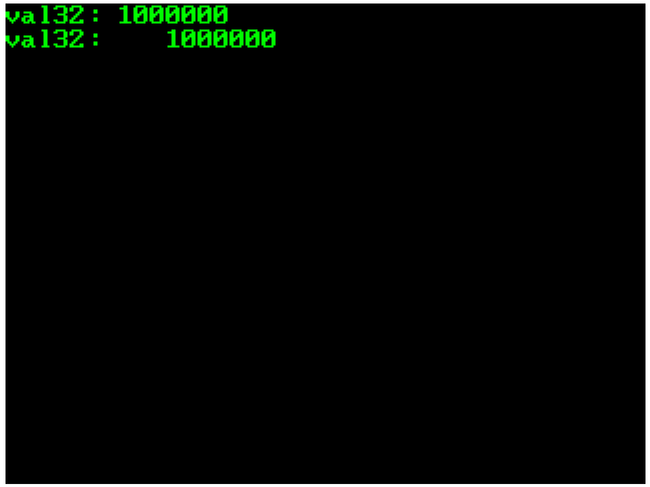
The 32-bit data found starting at address ***16***, in this example, is "**0x000F 4240**", the decimal equivalent of which is "**1000000**".



The Width and Zero Flag Sub-specifiers

The Width Sub-specifier

Consider the output below.



The code for the first line is

```
print("\n\n");
ptr := str_Ptr(val32);
print("val32: ");
str_Printf(&ptr, "%ld");
```

The code for the second line is

```
print("\n");
ptr := str_Ptr(val32);
print("val32: ");
str_Printf(&ptr, "%10ld");
```

Note that in the second line, the number has three spaces preceding it. This is because the width specifier was used in the ***str_Printf(…)*** function.

```
str_Printf(&ptr, "%10ld");
```

Here the width specifier value is ***10***, so the field width of the printed figure is ten digits, and since the number is only seven digits, it is preceded by three space characters.

The Zero Flag Sub-specifier

Suppose we want the number to be preceded with zeros rather than spaces, we would write,

```
print("\n");
ptr := str_Ptr(val32);
print("val32: ");
str_Printf(&ptr, "%010ld");
```

Here the width sub-specifier is preceded by the zero flag sub-specifier, which would cause the number to be left-padded with zeros instead of spaces. To illustrate,

Therefore, without the zero flag sub-specifier, the default character with which a number, printed with a certain field width, is to be left-padded is the space character. The width and zero flag sub-specifiers can be used with other format specifiers besides the long signed decimal format specifier.

The Designer project for the discussions on the long signed decimal format specifier and the width and zero flag sub-speficiers is "**stringsBasics6.4dg**" (attached).

Printing Negative Numbers

Print a Negative Long Decimal Number

To print a negative long decimal number, we first generate a negative 32-bit integer by multiplying the two 16-bit integers shown below.

```
var val32[2];
var ptr;

gfx_ScreenMode(LANDSCAPE) ;          // change manua

umul_1616(val32, 0xFF12, 0xFF34);
ptr := str_Ptr(val32);

print("ptr old: ",ptr,"\n");

print("val32: ");
str_Printf(&ptr, "%ld");

print("\n");
print("ptr new: ",ptr);
```

Again, the function ***umul_1616(…)*** performs an unsigned multiplication of two 16-bit values placing the 32-bit result in a two-word array. In this example, the two 16-bit values are ***0xFF12*** and ***0xFF34***. When treated as unsigned numbers and multiplied together, the product of these is **0xFE46 BDA8**. If we print the contents of the word array ***val32*** in hexadecimal format,

```
print("\n");
print("val32[0]: ",[HEX]val32[0]);

print("\n");
print("val32[1]: ",[HEX]val32[1]);
```

we get,

```
val32[0]: BDA8
val32[1]: FE46
```

Using the long signed decimal format specifier, "**%ld**", would cause ***str_Printf(…)*** to treat the two-word data as a signed 32-bit integer. Hence, we get the decimal equivalent value of "**-28918360**" for "**0xFE46 BDA8**".

```
var val32[2];
var ptr;

gfx_ScreenMode(LANDSCAPE) ;          // change manua

umul_1616(val32, 0xFF12, 0xFF34);
ptr := str_Ptr(val32);

print("ptr old: ",ptr,"\n");

print("val32: ");
str_Printf(&ptr, "%ld");

print("\n");
print("ptr new: ",ptr);
```

```
val32: -28918360
```

Print a Long Unsigned Decimal Number

To make ***str_Printf(…)*** treat the data inside ***val32[0]*** and ***val32[1]*** as an unsigned 32-bit integer, we use the long unsigned decimal format specifier "***%lu***".

```
// print a long unsigned decimal number
print("\n\n");
ptr := str_Ptr(val32);
print("val32: ");
str_Printf(&ptr, "%lu");
```

Hence we get the equivalent decimal value "**4266048936**".

```
val32: 4266048936
```

Dynamic Construction of the Format Specifier

As was shown in the application note Designer or ViSi 4DGL Strings Print Formats – the String and Character Format Specifiers, the format specifier argument of the ***str_Printf(…)*** function can also be a word-aligned string pointer, allowing dynamic construction of the printing format. We will now use dynamically constructed format specifiers, together with the width and zero flag sub-specifiers, to come up with the formatted display output shown below.

```
val32: -0028918360
val32: 04266048936
```

The code snippet for the above output is:

```
// print a long signed decimal number
print("\n\n");
print("val32: ");
to(format); print("%010ld");
ptr := str_Ptr(val32);
str_Printf(&ptr, format);

// print a long unsigned decimal number
print("\n");
print("val32: ");
to(format); print("%011lu");
ptr := str_Ptr(val32);
str_Printf(&ptr, format);
```

The Designer project for the remaining part of this application note is "**stringsBasics7b.4dg**" (attached). Although the examples are simple, the ability to construct a format specifier dynamically can be a powerful tool.

Run the Program

For instructions on how to save a **Designer** 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

Designer Getting Started - First Project

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

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

Proprietary Information

Disclaimer of Warranties & Limitation of Liability