APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi Displaying Texts And Strings on Goldelox Displays

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

Description

This application note is dedicated to providing the reader a simple and straight forward documentation about string functions. This application 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.

The tools needed include the following;

- Any of the following 4D Goldelox display modules:
  uOLED-96-G2
  uOLED-128-G2
  uOLED-160-G2
  uLCD-144-G2
  uTOLED-20-G2
  or any superseded module that supports the Designer and/or ViSi environments
- 4D Programming Cable or µUSB-PA5
- 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

This application note will simply teach the user how to display text and strings on a Goldelox display module. This will demonstrate a simplified approach in to learning text and string related functions of the 4DGL programming language used in Goldelox platform.

This application note focuses on a step-by-step approach in learning the Goldelox text and string functions. It is composed of a set of application notes all dedicated to discuss a common group or pair of functions of the Goldelox internal instructions.

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 Goldelox

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

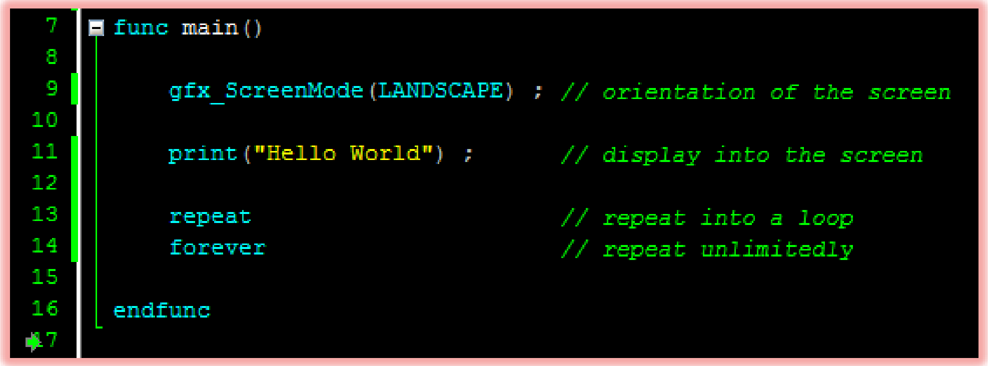
ViSi Getting Started - First Project for Goldelox

Design the Project

Using the print(…) Function

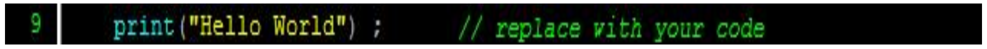
The print function of the 4DGL is a versatile function. It is one of the most useful and most commonly used function when it comes to directing the Goldelox processor to display a string, number, or values in several numerical formats unto the device's display.

A part of the program that is automatically displayed when starting a Designer environment based program includes an example on how to use the **print(…)** function. Let us begin understanding the **print(…)** function using the program below.

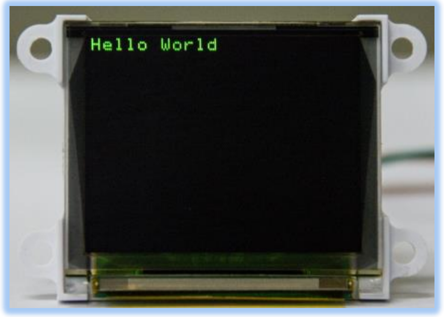


```
func main()

    gfx_ScreenMode(LANDSCAPE) ; // orientation of the screen

    print("Hello World") ;      // display into the screen

    repeat                      // repeat into a loop
    forever                     // repeat unlimitedly

endfunc
```

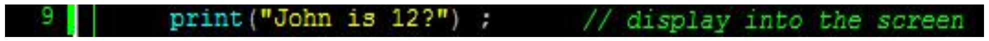
In this program, the print function is used to display into the Goldelox screen the words Hello World. Notice that is enclosed by a pair of quotation marks. The quotation marks are used to indicate the string to be displayed. If we are to run this program we shall notice that the text quotation marks are not displayed but only the string inside of it.



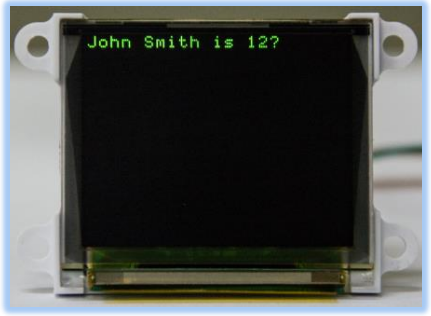
```
print("Hello World") ;      // replace with your code
```



This is how simple it is to use the **print(…)** function. Numbers, spaces and other special characters can also be displayed directly to the screen using the same program. So changing the content of the text in the **print(…)** function, for instance we write these information: "John Smith is 12?".

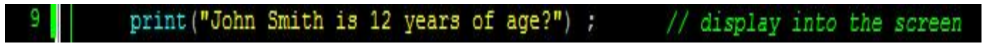


```
print("John is 12?") ;      // display into the screen
```

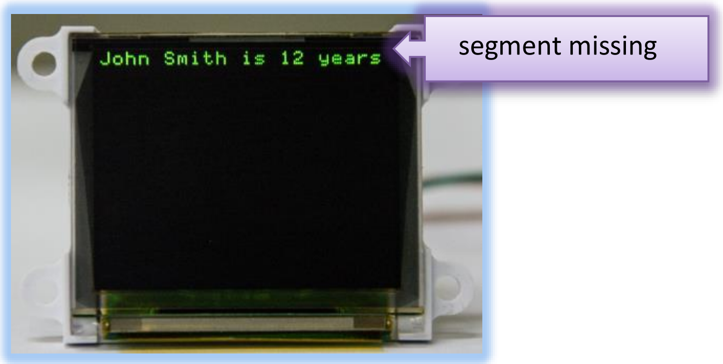


From all of the samples above, you will notice that anything that is written inside the quotation marks are directly written or displayed into the screen. If using the **print(…)** function, the display always starts at the topmost left of the screen when displaying a set of characters or phrases.

Moving another step forward, Let us try replacing the sentence again the phrases inside the quotation marks with a longer set of string: “John Smith is 12 years of age?”. After downloading this to the device, we will have an output similar to the image shown here.

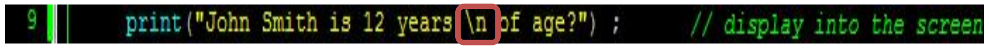


```
print("John Smith is 12 years of age?") ;        // display into the screen
```

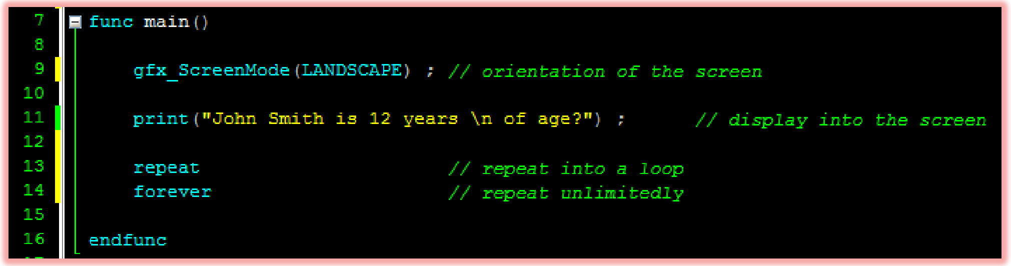


We can see that from the last string set in to the print function that a couple of words are missing after being displayed. The reason behind is that the print function directly sends out the data into the screen without any automatic text wrapping. There is a command that is very useful for these kinds of simple errors – the newline command **\n**.

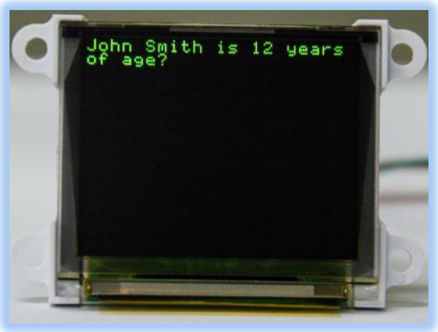
The newline command \n tells the print function- print the succeeding words of the string to a newline. This is now one simple way of formatting output by wrapping the extra words to the next line.



```
print("John Smith is 12 years \n of age?") ;        // display into the screen
```



```
func main()

    gfx_ScreenMode(LANDSCAPE) ; // orientation of the screen

    print("John Smith is 12 years \n of age?") ;      // display into the screen

    repeat                       // repeat into a loop
    forever                      // repeat unlimitedly

endfunc
```

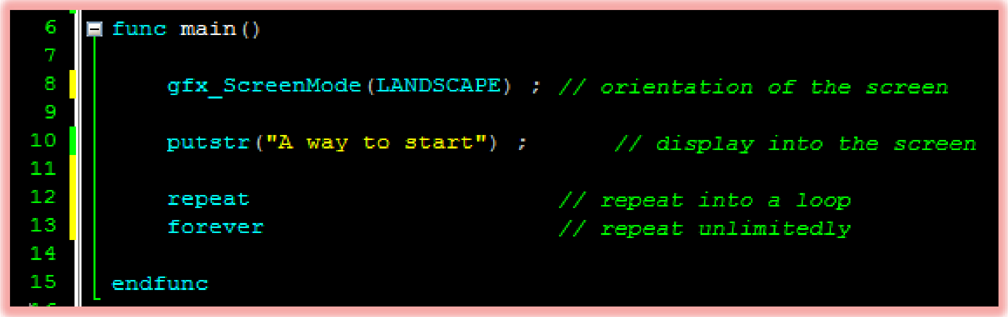


The **print(…)** function is relatively very simple to use and that the output can be readily seen after downloading the program to the Goldelox display module. At this point we can say that regardless of the character being a letter in the alphabet or a number the print(…) function can readily have this sent and displayed to the screen of the Goldelox display module.

Using the putstr(…) function

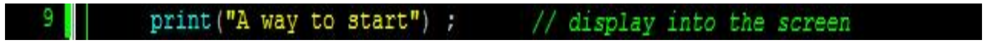
Printing a set of character into the display screen is relatively easy. Another useful command that can be used to achieve this purpose is the put string function which has a syntax **putstr(…)**.

Referring to the program below we have the **putstr(…)** function to direct the processor to display the *Hello World* into the screen display. Let us look at the statement written in line 10.



```
 6  func main()
 7
 8      gfx_ScreenMode(LANDSCAPE) ; // orientation of the screen
 9
10      putstr("A way to start") ;      // display into the screen
11
12      repeat                      // repeat into a loop
13      forever                     // repeat unlimitedly
14
15  endfunc
```

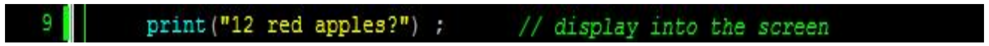
The need for the quotation marks to enclose the string to be displayed is a requirement of this function.



```
9      print("A way to start") ;      // display into the screen
```

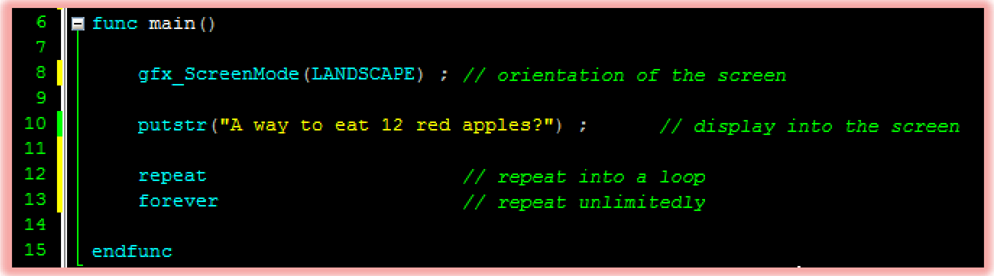
The characters that can be included to the string could again be a letter, a number of special characters. Illustrating this, let us change the text to be displayed with – “12 red apples?”.

Using the **putstr (…)** function would look like this and the result will be similar to the image below.



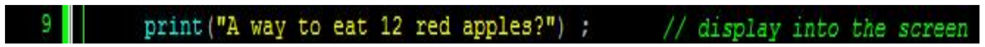
```
9      print("12 red apples?") ;      // display into the screen
```

It can be noticed that the manner of using the **putstr(…)** is the same as that of the print(…). The display processor translates this functions and directly outputs the string to the screen. Now we take another example program that shall again display a sentence. This program uses the **putstr(…)** function to direct the processor again to simply display the message to the display module.

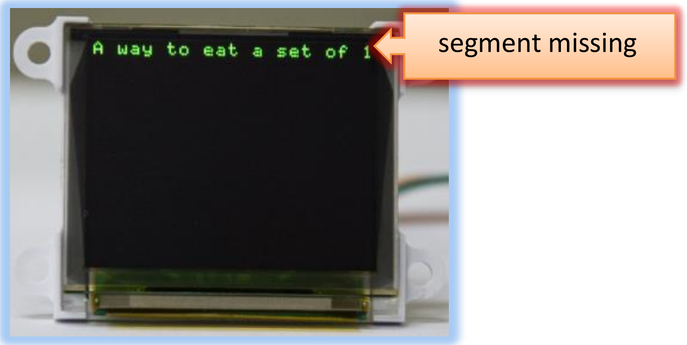


```
func main()

    gfx_ScreenMode(LANDSCAPE) ; // orientation of the screen

    putstr("A way to eat 12 red apples?") ;      // display into the screen

    repeat                          // repeat into a loop
    forever                         // repeat unlimitedly

endfunc
```

With the use of the **putstr(...)** function, the text can be clearly displayed into the display module. On the otherhand, due to the length of the number of the characters contained in the function, several of the remaining character succeeding the numerical number 1 cannot be displayed.

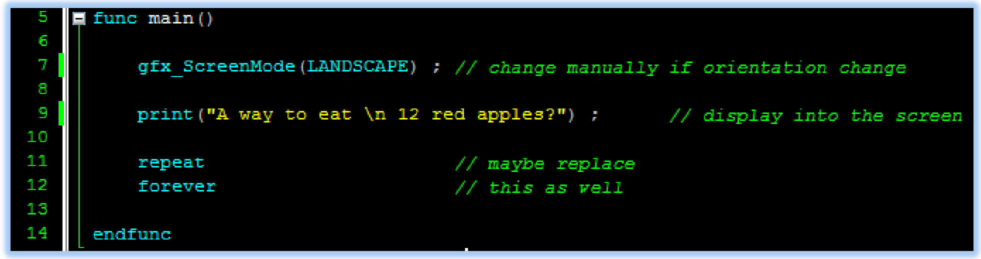


```
    print("A way to eat 12 red apples?") ;      // display into the screen
```

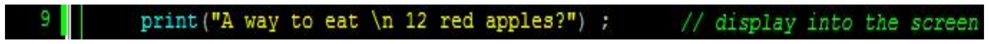


Similar to the result using the print(...) function, the characters which exceed the limits of the horizontal character capacity of the screen are neither displayed nor move to the next line. The **putstr(...)** function directs the processor to continue displaying without any additional formatting.

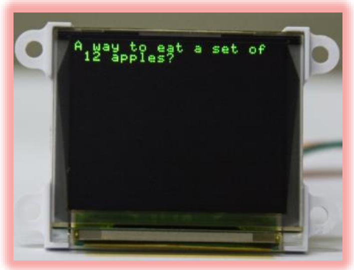
Similar to print(...) function, the addition of a newline command is very useful in correcting this kinds of display problems. Adding this to the **putstr(...)** have an output the same as the one below.



```
func main()

    gfx_ScreenMode(LANDSCAPE) ; // change manually if orientation change

    print("A way to eat \n 12 red apples?") ;      // display into the screen

    repeat                          // maybe replace
    forever                         // this as well

endfunc
```



```
    print("A way to eat \n 12 red apples?") ;      // display into the screen
```

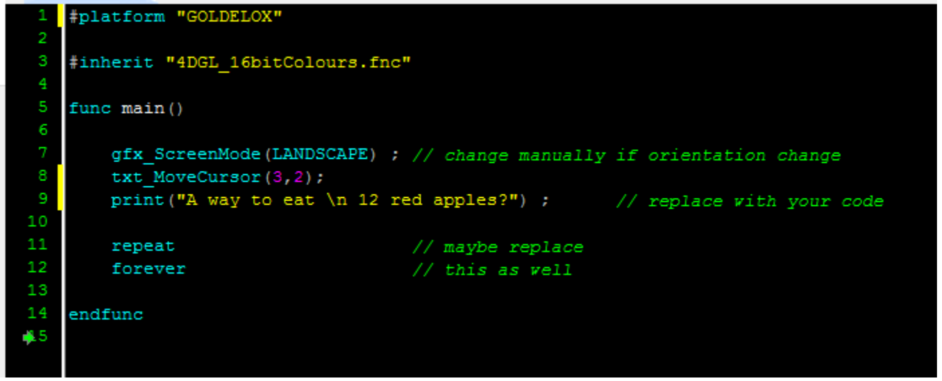


At this point, we are now able to simply display strings in our display module using the **print(...)** and **putstr(...)** functions. A useful exercise at this part is to try and put the \n at any point inside the string. Observe and take note of the effects of placing it at different parts of string.

Change the Point of Origin

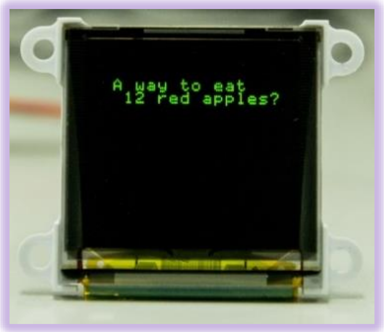
The origin point of text and string printing can be changed. Using the 4DGL Goldelox internal function - txt_MoveCursor(), printing at any point of the screen is possible.

This function includes two parameters for it to function properly. A line and column data is written inside the parentheses. Note that this function must precede the print() or putstr() function statement. Below is a program that shows how to use this function.



```
#platform "GOLDELOX"

#inherit "4DGL_16bitColours.fnc"

func main()

    gfx_ScreenMode(LANDSCAPE) ; // change manually if orientation change
    txt_MoveCursor(3,2);
    print("A way to eat \n 12 red apples?") ;      // replace with your code

    repeat                          // maybe replace
    forever                         // this as well

endfunc
```

The txt_MoveCursor(3,2) statement, means that the succeeding printing function will start to display text and strings on the 2nd column of the 3rd line. Downloading this program to the Goldelox display module will have a display output similar to the image below.



```
  A way to eat
 12 red apples?
```

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 (this option is not available for Goldelox displays), 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 Goldelox

The uOLED-96-G2 and/or the uOLED-160-G2 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.