APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi Word Addressing Architecture

DOCUMENT DATE:	**12th APRIL 2019**
DOCUMENT REVISION:	**1.1**

WWW.4DSYSTEMS.COM.AU

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 4D Workshop 4 – DESIGNER environment. The tools needed includes 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  or any superseded module that supports the ViSi environment
- 4D Programming Cable / uUSB-PA5/uUSB-PA5-II
- 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.
- This application note requires that the reader has a basic knowledge of any programming language such as C.

Content

Application Overview

This Application Note is solely intended to discussing the Word Addressing based Underlying Architecture. It includes a detailed explanation and example of the ways to saving strings into a variable. It also encompasses examples that will help the reader understand the way or manner by which string data are saved.

The underlying architecture of the GOLDELOX processor is a 2-byte word, this extends to the addresses used to access memory. This means that for strings, normal addresses and pointers cannot be used. Strings need to be addressed in 8 bit increments. The GOLDELOX processor is capable of string handling through user defined functions. These functions can be programmed to perform a certain purpose, for example, find and match characters from a string or to perform concatenation of strings.

Before continuing with this application note, it is very important that the user already knows how to use the Workshop IDE - Designer environment and that the user is able to understand and use the basic text and string functions of the GOLDELOX 4DGL 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

Storing string data into a variable array

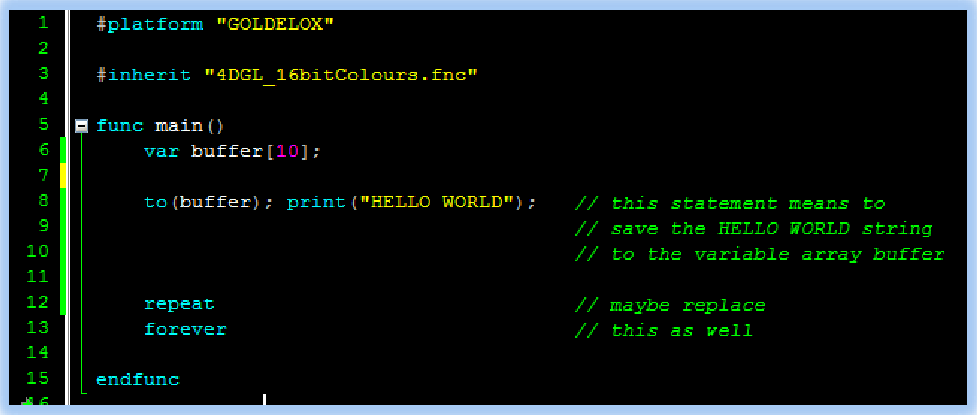
This section includes a several subtopic that discusses significant process and information that is related to storing of string data and how the data address are determined.

How to save a string into a variable array?

Saving a string into a variable array is relatively easy. We save string data into a variable by writing the string itself to a declared variable array.

The *to(outstream)* commands directs the processor to a media or to a variable array alongside with the putstr(…) and print(…) will result to a statement that is capable of attaining this save string process. For a better visualization of this process let us use the following example.

We are going to save the string '*HELLO WORLD*' into a variable array named buffer. Putting this into a program would look like this:



```
#platform "GOLDELOX"

#inherit "4DGL_16bitColours.fnc"

func main()
    var buffer[10];

    to(buffer); print("HELLO WORLD");   // this statement means to
                                        // save the HELLO WORLD string
                                        // to the variable array buffer

    repeat                              // maybe replace
    forever                             // this as well

endfunc
```

In the above program a variable buffer with an array size of 10 is being declared. The size declaration of the variable array means that there are 10 indexes. In turn each of the index contains a pair of bytes. It may simply be stated that, for this example, each of the array index shall contain 2 characters. Therefore, the total bytes that can be held in an array of this size would be 20.

Character/byte capacity = array size X 2

How are the string data grouped in a buffer index?

Next at hand is to understand how these characters are lumped/grouped in pairs. Putting this into a pairing figure it shall look like this :

index	0	1	2	3	4	5
character	H / E	L / L	O / space	W / O	R / L	D / none

The figure above only shows the pairing of the characters in the string. Notice that the space is included with the set of the characters.

If for example, we change the string to ‘GOOD DAY’, then the indexes of the array will have this pairing of character.

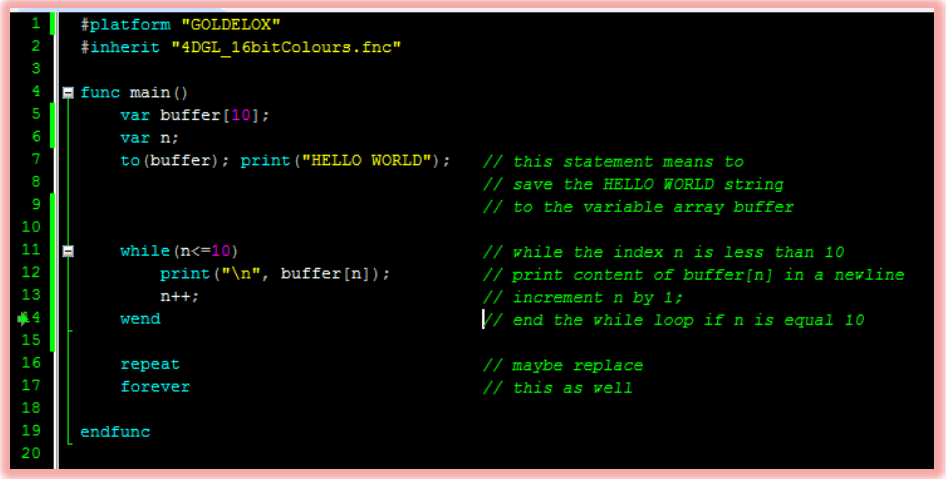
index	0	1	2	4
character	G / O	O / D	Space / D	A / Y

Aside from the grouping of the characters, one very important part to understand is the way these are saved into the array. Let us take the ‘GOOD DAY’ example to picture the result when saving a string data to an array.
The equivalent values of the characters

index	0	1	2	3	4	5
character	H / E	L / L	O / space	W / O	R / L	D / none
HEX	48/45	4C/4C	4F/20	57/4F	52/4C	44/none

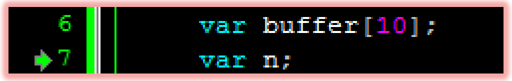
A simple program to illustrate saving of a string

For us to have a better look on the way the characters are saved in the variable array buffer, let’s take this example program that will save “HELLO WORLD” into the variable array buffer. After saving the string we will print the content of the variable array while indicating the index number and the content.



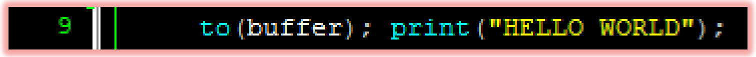
```
1   #platform "GOLDELOX"
2   #inherit "4DGL_16bitColours.fnc"
3
4   func main()
5       var buffer[10];
6       var n;
7       to(buffer); print("HELLO WORLD");   // this statement means to
8                                           // save the HELLO WORLD string
9                                           // to the variable array buffer
10
11      while(n<=10)                        // while the index n is less than 10
12          print("\n", buffer[n]);         // print content of buffer[n] in a newline
13          n++;                            // increment n by 1;
14      wend                                // end the while loop if n is equal 10
15
16      repeat                              // maybe replace
17      forever                             // this as well
18
19  endfunc
20
```

Before looking into the result of the above program, let us first look into the details. The program started out with the declaration of the variable array buffer with a sized of 10 pairs of bytes.



```
6       var buffer[10];
7       var n;
```

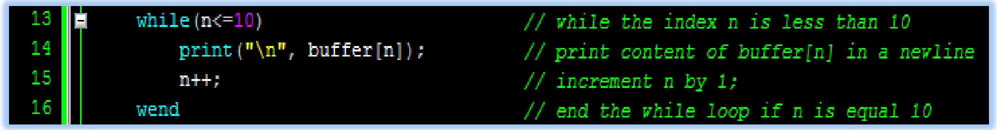
We have also declared a variable *n* that will serve as the representation of our index positions. After the declaration of the variable buffer and its size, the ‘HELLO WORLD’ string is being stored to the array *buffer.*



```
9       to(buffer); print("HELLO WORLD");
```

In the statement written in line 9, the variable *to(…)* function pointed to the variable *buffer* as the address where the string data ‘HELLO WORLD’ will be written. This also means that the hexadecimal values of the string data is being placed into the indices of *buffer* in pairs.

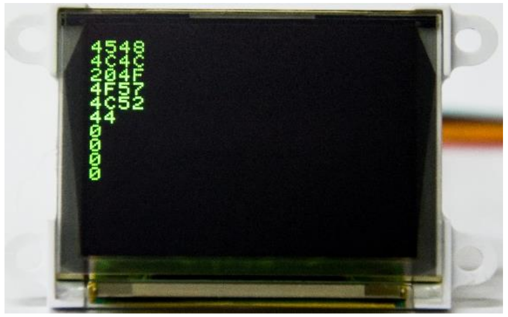
The next part is the *while(…)/wend* loop. The while loop simplifies the need for repetitive statements that will print the content of the buffer index n.



```
while(n<=10)                        // while the index n is less than 10
    print("\n", buffer[n]);         // print content of buffer[n] in a newline
    n++;                            // increment n by 1;
wend                                // end the while loop if n is equal 10
```

This statement implies that the printing of the content of the buffer is according to the index - n. The starting index count is zero and continuously incremented with the n++ statement. Every time that the index count n is incremented the value of the buffer[n] is displayed on a newline. The newline is given by the \n command. This while-loop will continue until the value of n will be equal to 10.

After compiling this program and downloading it to the GOLDELOX display module, the result will be a set of 2 bytes which will be displayed one index at a time.



```
4548
4C4C
204F
4F57
4C52
44
0
0
0
0
```

How the string data are arranged inside the buffer?

We have observed from the preceding section that the hex values of the characters are not arranged the same way they are written in the string. We know that an index of a buffer can hold two bytes – one of which may be pertained to as a higher byte and a lower byte.

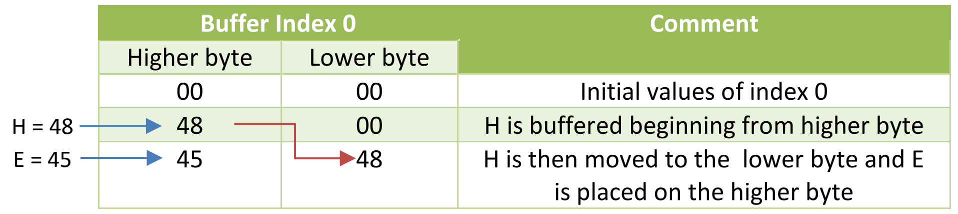
When the processor is saving the string data, it first fills-out the higher bit before the lower bit. After filling the first index, this index is then incremented and the process is repeated again until all of the string data is saved. Illustratively this can be represented as:

Variable Array Buffer (H= higher byte; L= lower byte)												
	Index 0		Index 1		Index 2		Index 3		Index 4		Index 5	
	H	L	H	L	H	L	H	L	H	L	H	L
Hex	**45**	**48**	**4C**	**4C**	**20**	**4F**	**4F**	**57**	**4C**	**52**	**44**	**00**
Character	E	H	L	L	space	O	O	W	L	R	D	END

To further explain the process, consider the first two letters of the string - H and E. When the string was saved in to the buffer array, the letter H fills the higher byte of the index.

When it is the turn of the letter E to be saved, the first letter H is moved to the lower byte position thus giving space for E to be placed in to the higher byte position.

Referring to the transition table of the process:



	Buffer Index 0		Comment
	Higher byte	Lower byte	
	00	00	Initial values of index 0
H = 48	48	00	H is buffered beginning from higher byte
E = 45	45	48	H is then moved to the lower byte and E is placed on the higher byte

This process is repeated until all the characters in the array are saved or until the declared array variable has space to save the characters. After all the characters have been saved, the string data is terminated with a value equal to 0.

Summary

The GOLDELOX architecture requires that the string data stored on a buffer must be addressed in 8 bit increments. The normal addressing and pointers must follow the required increments to be able to access correct data. Knowing this underlying architecture will help the user in the creation of user-defined string handling functions.

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 uLCD-32PTU, uLCD-35DT, uOLED-96-G2, and/or uOLED-160-G2 display modules are commonly used as examples, but the procedure is the same for other displays.