APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Displaying Third Party Fonts

FAT16

DOCUMENT DATE: **21st JANUARY 2019**
DOCUMENT REVISION: **1.1**

Description

This application note explains how custom fonts can be used on a Picaso or Diablo16 display module in the ViSi environment:

Before getting started, the following are required:

- Any of the following 4D Picaso and gen4 Picaso display modules:

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

  and other superseded modules which support the ViSi Genie environment

- 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. The display module used in this application note is the uLCD-32PTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.
- 4D Programming Cable / uUSB-PA5/uUSB-PA5-II
  for non-gen4 displays(uLCD-xxx)
- 4D Programming Cable & gen4-PA, / gen4-IB / 4D-UPA
  for gen4 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

There are three built-in fonts of the Picaso processor. These are:

- **Font1** (5x7)
- **Font2** (8x8)
- **Font3** (8x12)

The user might need more stylish and larger size fonts which is a need addressed in this application. The user can also import ANSII or UNICODE fonts. For the Diablo16 processor, the available system font IDs are:

- 1 for FONT_1 = System 5x7
- 2 for FONT_2 = System 8x8
- 3 for FONT_3 = System 8x12 (Default)
- 4 for FONT_4 = System 12x16
- 5 for FONT_5 = MS San Serif 8x12
- 6 for FONT_6 = Deja Vu Sans Condensed 9pt
- 7 for FONT_7 = Deja Vu Sans 9pt
- 8 for FONT_8 = Deja Vu Sans Bold 9pt
- 9 for FONT_9 = System 3x6
- 10 – Not currently available for SPE Serial, N/A
- 11 for FONT_11 = EGA 8x12 font

A customer might need the external fonts to be displayed in two ways,

- Display some text set in the design time.
- Display the test in run time.

Both of these are achievable.

When you set the text in design time, the string will be stored on the uSD card and can be displayed whenever it's needed. Whereas in run time a text could be displayed using putch(), putstr(), print(), or putnum() command.

NOTE: The file_Dir() command is a command that writes the list of directory directly to the screen. This command is also affected by the fonts change.

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

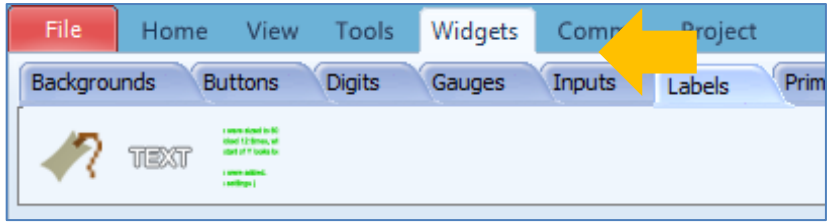
For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

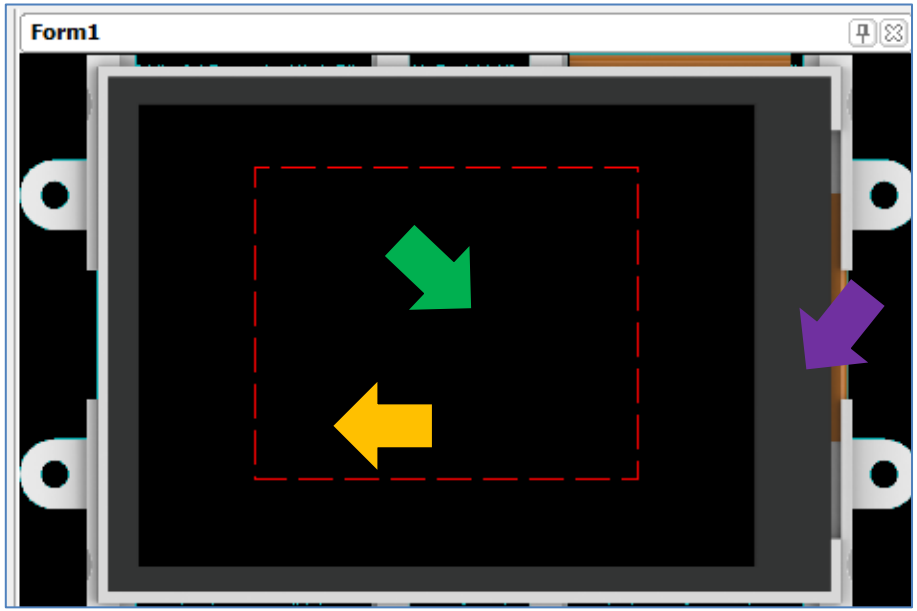
Design the Project

Add a String Object

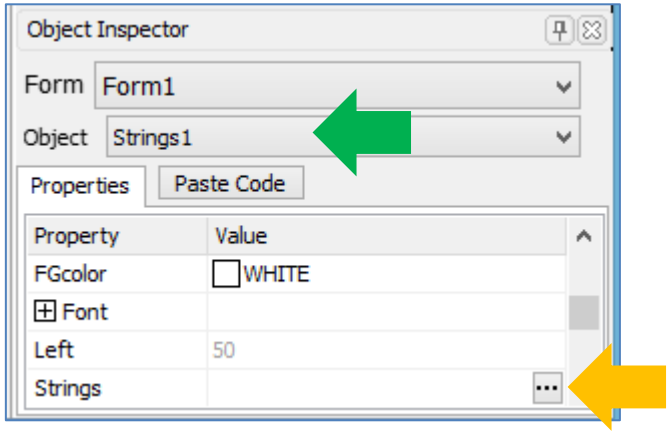
Go to **Widgets**, select **Strings** object under the **Labels** tab.



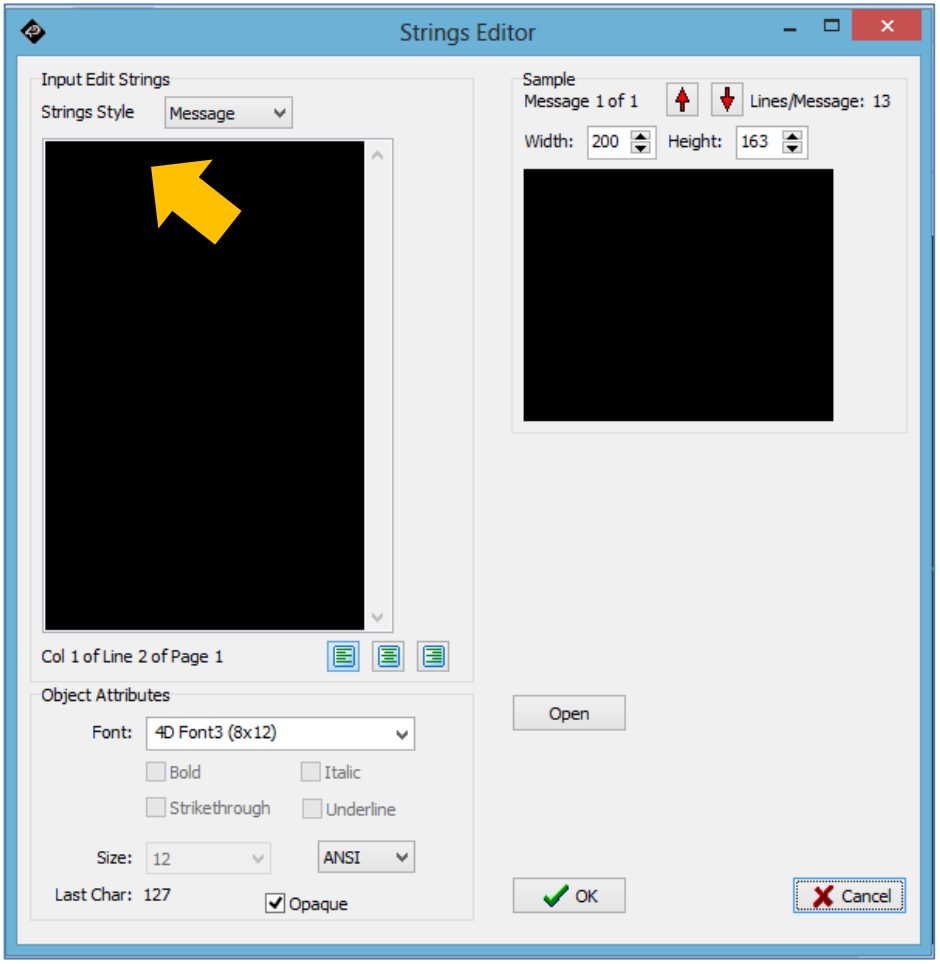
Click on to the WYSIWYG screen to drop the string object. A string object has now been created. This is Strings1.



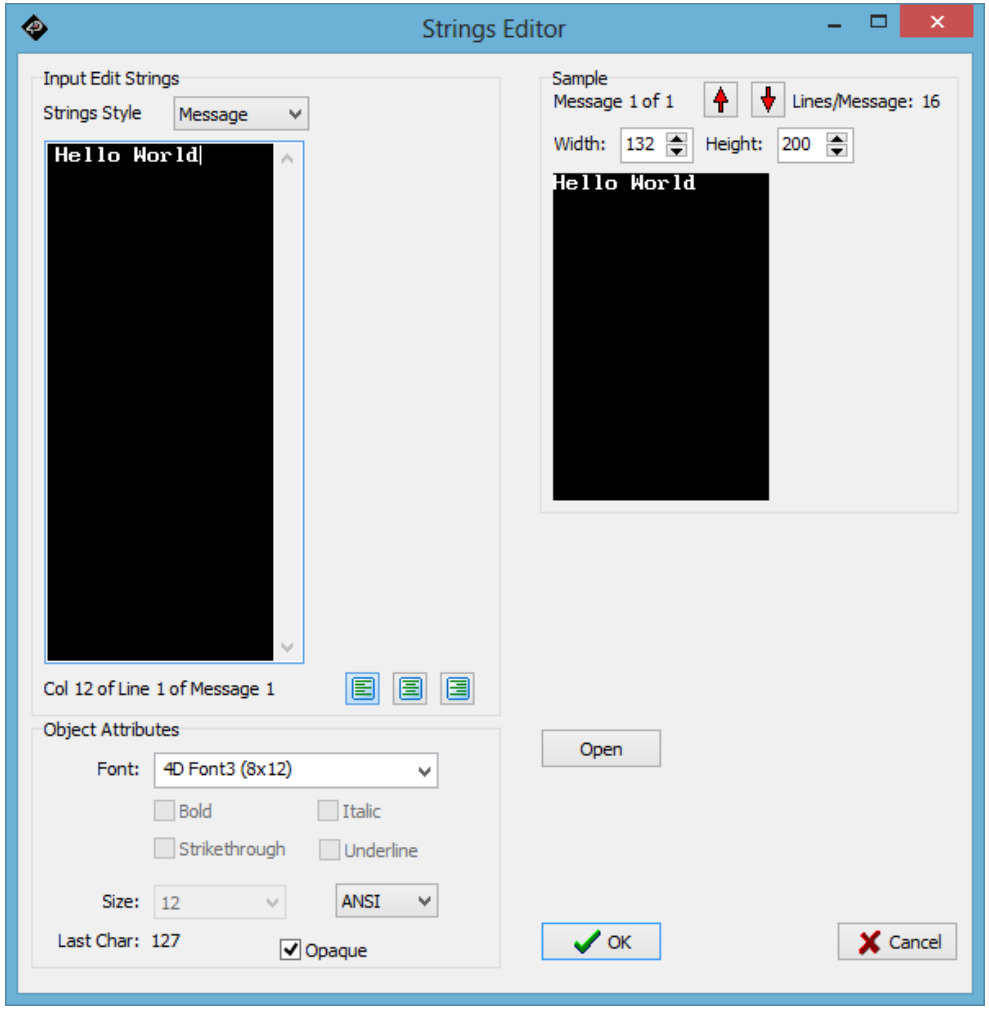
Click on the ellipsis dots of the **Strings** property of the Object Inspector for **Strings1**.



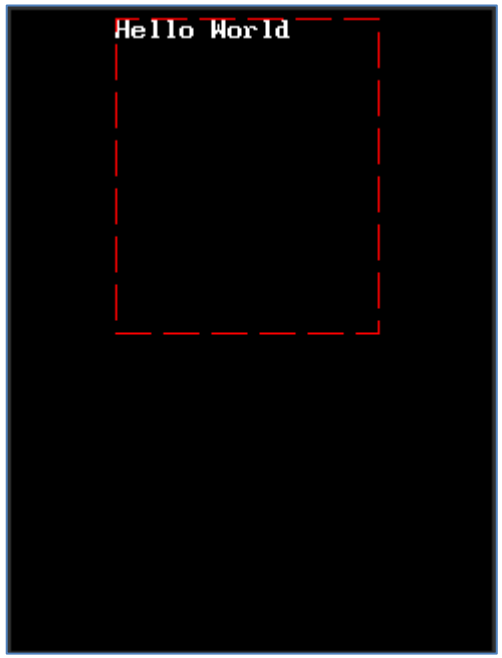
The Strings Editor window appears.



Click on the left window and input the string “Hello World”.



The output appears at the right window. Click OK. The WYSIWYG screen is now updated.



The string object can be moved to a new location and its area can be resized.

uSD Card Files for String Objects

When you compile your project, Workshop combines the contents of the string objects that you have added to the WYSIWYG screen into a single file. This file will then be copied to the uSD card, which will then be mounted to the display module. When the program runs on the display module, it will access this file to display any of the stored strings. For the object Strings1 that we have just created, the contained string is “Hello World”. Discussions of the functions used for opening and accessing the contents of a string file now follow.

Open a String File and Print a String

The filename extension of a string file is “**.txf**”. It is important for the program to know where the string of a certain string object is stored in the file. Workshop stores the locations (or memory addresses) of strings and other information in automatically-generated constants.

Open a String File

Going back to the object Strings1, the string “Hello World” will be added, when we compile the project, to a file with the extension “**.txf**”. For example,

Workshop derives the filename of a string file from that of your project. To retrieve the stored string, we must first open the file. The 4DGL function for opening a file is

```
hstrings := file_Open("filename.txf", 'r') ;
```

This function returns a handle, which can be used for further operations on the file.

Print a String

The function for printing a specific string from the uSD card is

```
PrintDisk(hstrings, Strings1StartH,
          Strings1StartL, Strings1Size, 0) ;
```

The first parameter for the function “**PrintDisk()**” is the handle for a string file that has been opened. Here the handle is “hstrings”.

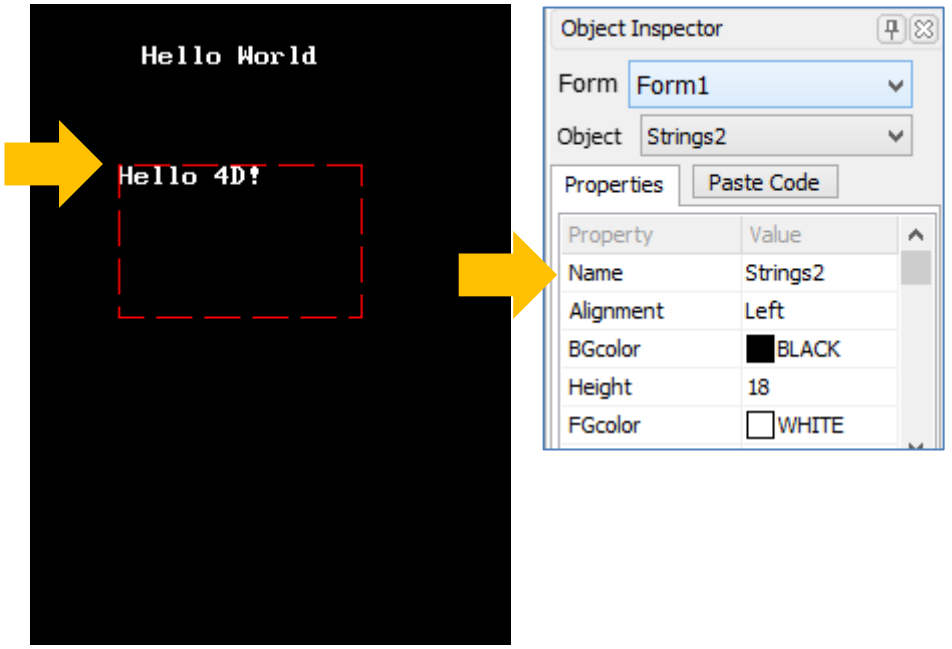
Memory Address of a String

The second and third parameters are the high and low words of the starting memory address of the string to be printed. The fourth parameter is the size of the string. The values of these parameters are taken care of for you by the ViSi environment. All that you will have to take note of is the integer inserted into the parameter names, which are actually constants. To illustrate,

```
PrintDisk(hstrings, Strings1StartH,
          Strings1StartL, Strings1Size, 0) ;
```

Index of a String Object

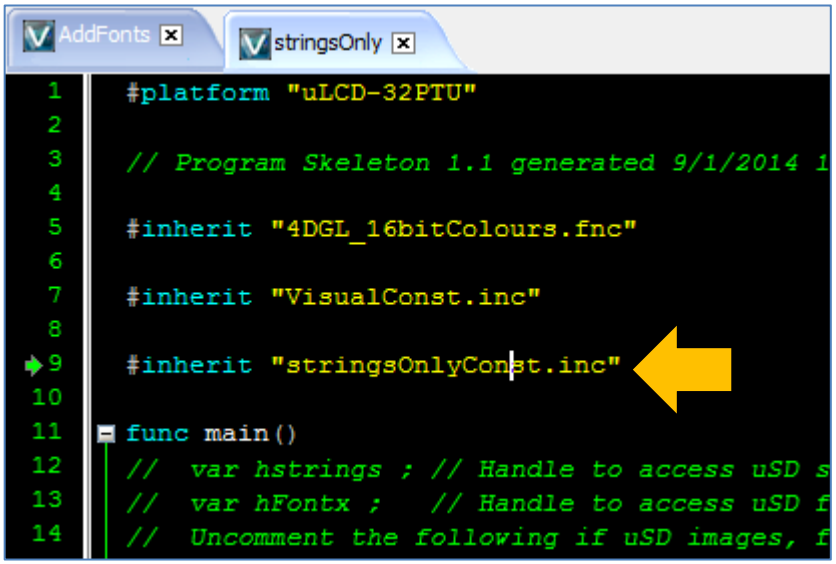
The integer corresponds to the index of the string object which contains the string to be printed. The function above therefore prints the contents of the object Strings1. Suppose we add another string object to the WYSIWYG screen. This would be Strings2.



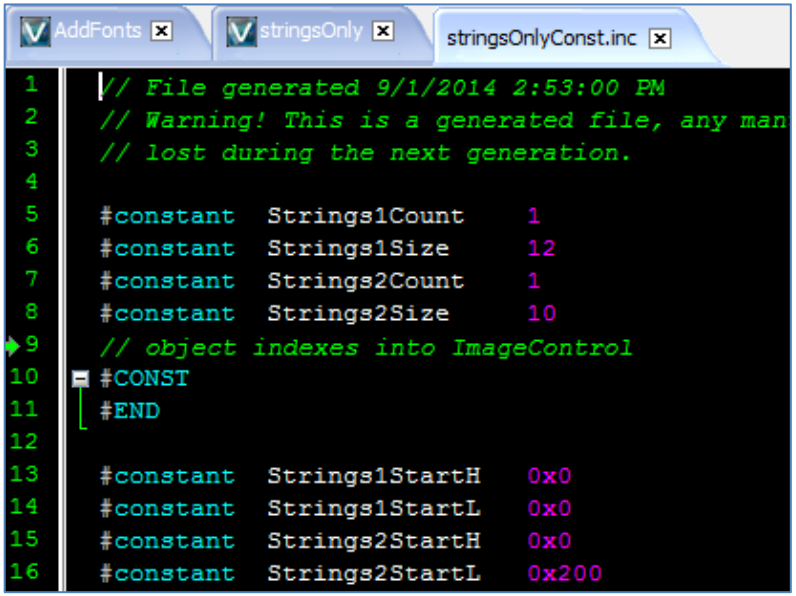
The correct parameters for printing the contents of this object would be

```
PrintDisk(hstrings, Strings2StartH,
          Strings2StartL, Strings2Size, 0) ;
```

To view the actual values of the constants used as parameters, open the include file indicated in the image below by putting the cursor on the filename text, clicking on the right mouse button, then selecting “Open file at Cursor”. Worskhop derives the filename of this include file from that of your project. Here the project was saved with the name “stringsOnly”.



The include file now opens. This include file contains constants and their values automatically generated by Workshop.



Note that the include file "stringsOnlyConst.inc" will only be generated after the project is saved and compiled.

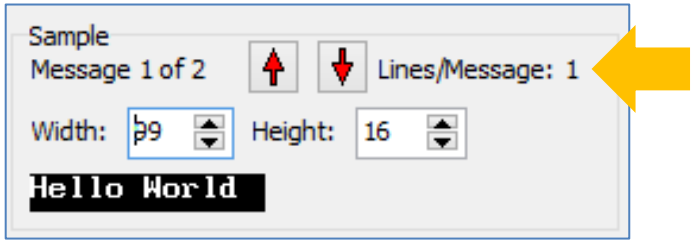
Message ID

The fourth parameter of the function "**PrintDisk()**" is the message ID.

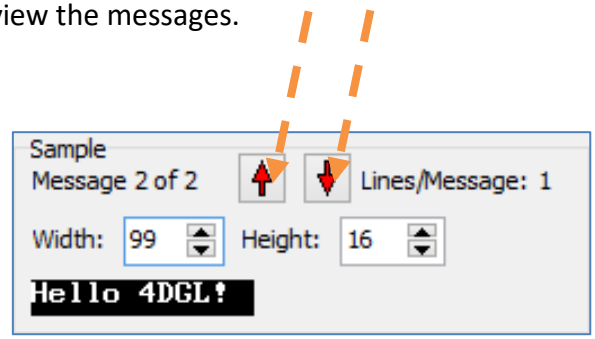
```
PrintDisk(hstrings, Strings1StartH,
                    Strings1StartL, Strings1Size, 0) ;
```

A string object can be edited so that it contains multiple messages. We can then print a specific message found inside a string object by setting the value of the fifth parameter of the function "**PrintDisk()**". Going back to the Strings Editor for Strings1, we add another line of text.

On the right part of the Strings Editor window it says “Lines/Message: 1”.



Since there are two lines of text added to the Input Edit Strings box, there are two messages therefore in the object Strings1. Click on the up and down arrows to preview the messages.



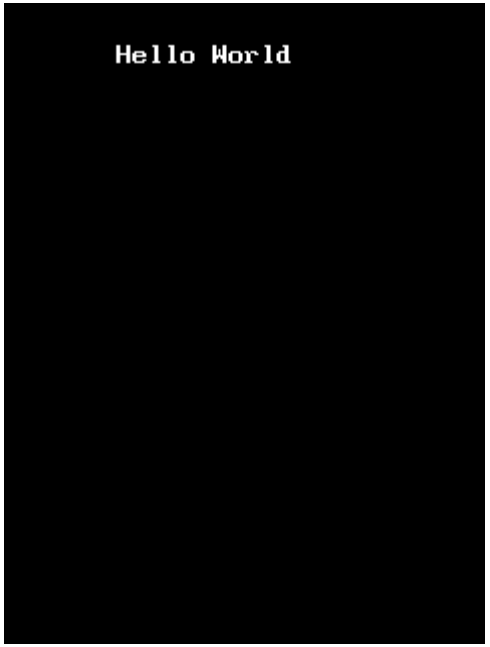
Experiment with the width and height of a string object to change the value of the property “Lines/Message”.

Print a Specific Message

The command for printing the first message of String1 is

```
PrintDisk(hstrings, Strings1StartH,
          Strings1StartL, Strings1Size, 0) ;
```

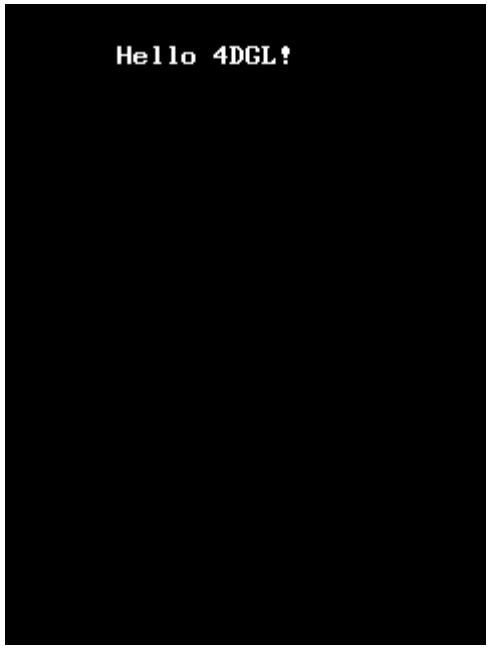
The output is



The command for printing the second message of String1 is

```
PrintDisk(hstrings, Strings1StartH,
          Strings1StartL, Strings1Size, 1) ;
```

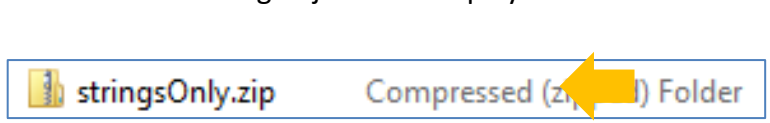
The output is



Note: The function "**PrintDisk()**" is defined in the include file "**PrintDisk.inc**". This file is not included in a newly created ViSi code by default, so you will have to include it manually. Here it is shown how it is included on line 10 of the 4DGL code.

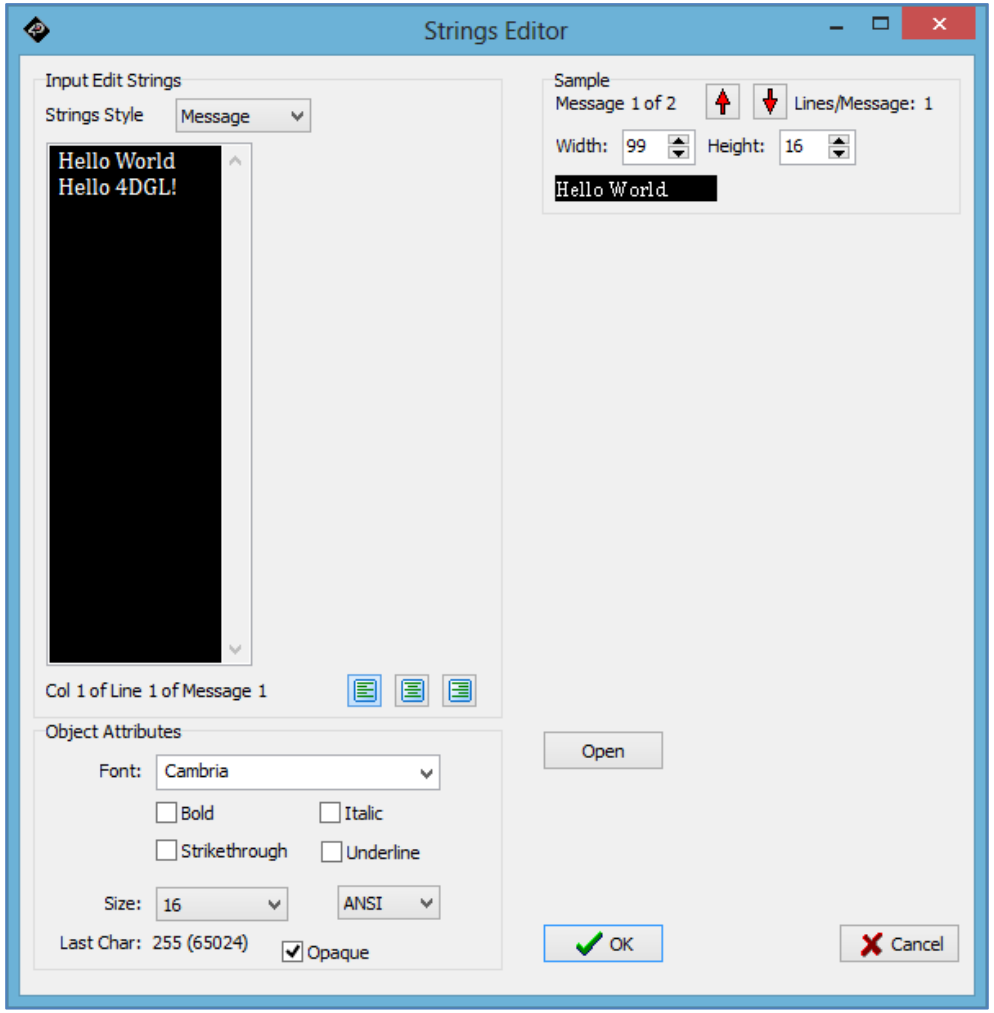
```
1  #platform "uLCD-32PTU"
2
3  // Program Skeleton 1.1 generated 8/28/2014
4
5  #inherit "4DGL_16bitColours.fnc"
6
7  #inherit "VisualConst.inc"
8
9  #inherit "NoName1Const.inc"
10 #inherit "PrintDisk.inc"
11 func main()
12 //  var hstrings ; // Handle to access uSD
13 //  var hFontx ;   // Handle to access uSD
14 //  Uncomment the following if uSD images,
15 /*
16     putstr("Mounting...\n");
17     if (!(disk:=file_Mount()))
18         while(!(disk :=file_Mount()))
19             putstr("Drive not mounted...");
20             pause(200);
21             gfx_Cls();
```

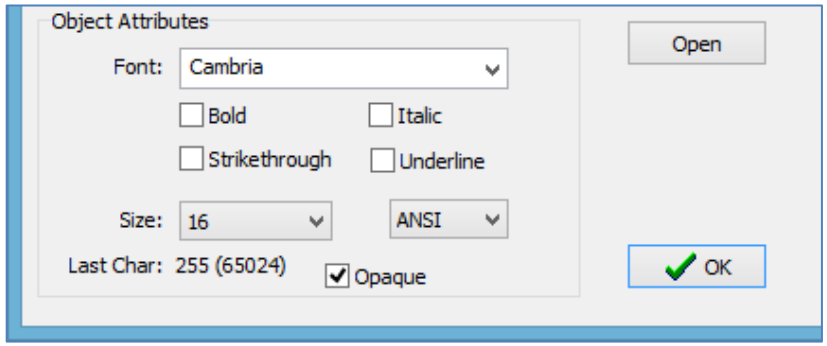
Attached is a zip file containing a simple project that shows how a string file is opened and how string objects are displayed.



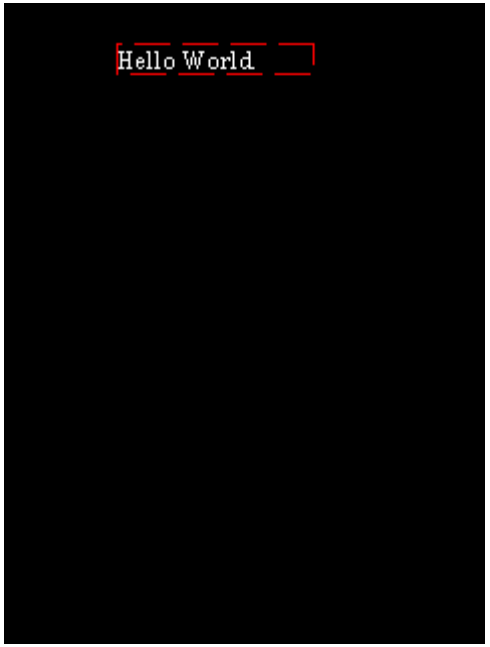
Set the Font

The font of a string object can be changed in the Strings Editor window. Going back to the object Strings1, set the values of the font and size properties as shown below. Click OK.





The string object on the WYSIWYG screen is now updated with a new font.

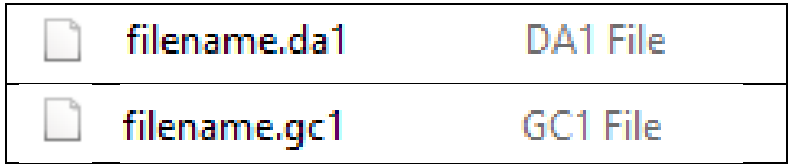


uSD Card Files for Fonts

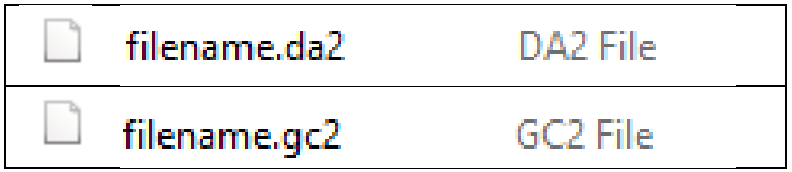
When you compile your project, Workshop generates the associated files for the fonts that you have added to the WYSIWYG screen. These files will be copied to the uSD card, which will then be mounted to the display module. When the program runs on the display module, it will access these files to print text onscreen using the desired font. The string may come from the uSD card, from the program memory, or it can be a literal constant. For our working string object, Strings1, the string is "Hello World" and the font is Cambria. Discussions of the functions used for loading files associated to a font now follow.

Generated Font Files

There are two files, generated by Workshop, associated to a certain font. These files have the filename extensions **".gcn**" and "**.dan**", where "**n**" refers to the index of a font added when designing using the WYSIWYG screen. Workshop will therefore generate the files below for the first font added to the project.



The second font added to the project will be associated with the files:



Similar to string files, Workshop derives the filename of font files from that of your project.

Load the Files for a Font

To use a font from the uSD card, we must first load it. The function for loading the two files associated to a font is

```
hFont1 := file_LoadImageControl
          ("filename.da1", "filename.gc1", 1);
```

This function returns a handle for the uSD card font. The handle can then be used for setting the current font ID. Here the font files are those associated to the first font added to the WYSIWYG screen.

Set the Font ID

The function

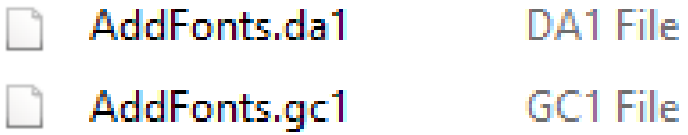
```
txt_FontID(fontID);
```

sets the current font. The parameter "fontID" can be a handle for a uSD card font that has been opened. To illustrate,

```
txt_FontID(hFont1) ;
```

This will set the current font to the first font that was added to the project. The succeeding print commands will now use this font until a new font ID is selected.

Now suppose we have selected the font "Forte" as the first font and have saved the project with the filename "AddFonts", Workshop would now generate the two files below for the font "Forte".



We would now load these files with the command

```
hFont1 := file_LoadImageControl
          ("AddFonts.da1", "AddFonts.gc1", 1);
```

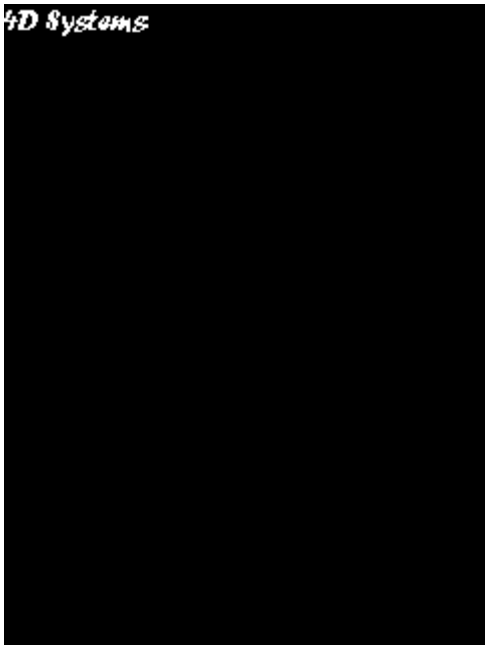
We would set the current font with the command

```
txt_FontID(hFont1) ;
```

And the commands

```
txt_FontID(hFont1) ;
print("4D Systems");
```

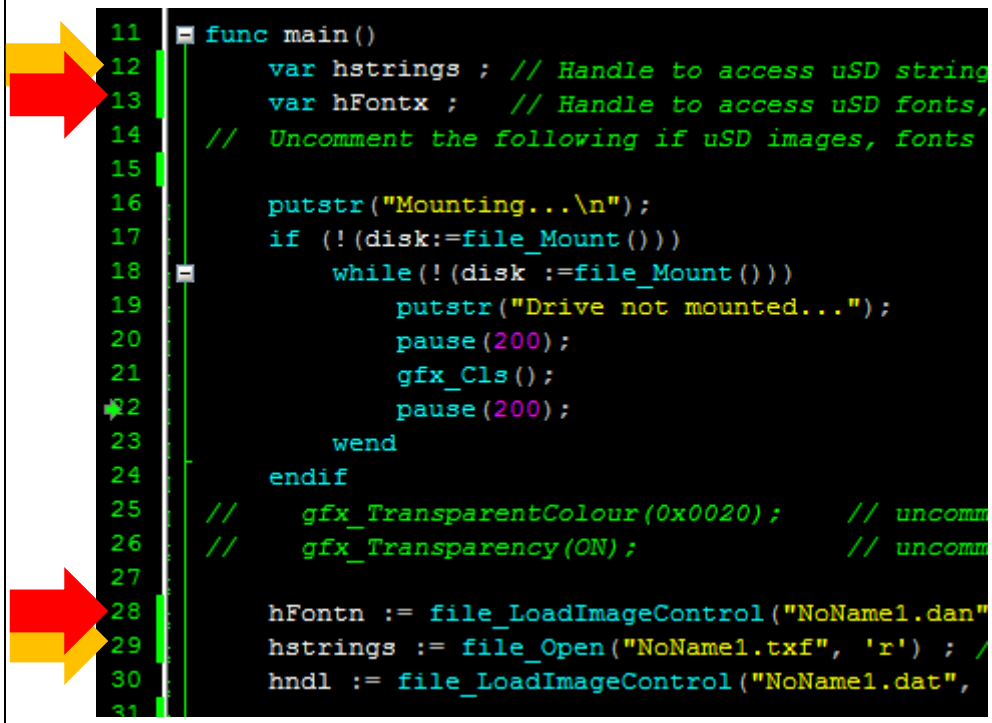
would produce an output similar to that shown below.



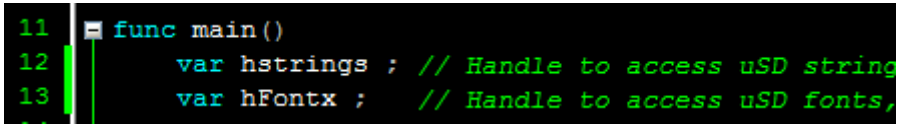
Built-in fonts can also be used as font IDs. Refer to the lists given in the **Application Overview** section of this application note. Attached is a simple program that demonstrates how external fonts or fonts from the uSD card are used.

Design with the Program Skeleton

With the foregoing separate discussions on the format of files associated to string objects and fonts and the functions used for loading and accessing these files, we can now proceed to modifying the default program skeleton of a newly created ViSi project. Note that several of the functions we discussed earlier are already a part of the program skeleton. We will just have to uncomment the lines with which we are interested. Indicated in the image below are the lines that will be needed in this application. The single-line comment symbols are removed. Also, the block comment symbols "/*" and "*/" on lines 15 and 31 are omitted as the contained block is also needed.



The variables "hstrings" and "hFontx" declared on lines 12 and 13 will be used as handles for a string file and font files, respectively.



The variable “hstrings” will be used as the handle for the string file “NoName1.txf”.

```
29      hstrings := file_Open("NoName1.txf", 'r') ; /
```

Again, Workshop will rename the text file when the project is saved.
The variable “hFontx” will be used as the handle for the files associated to a font.

```
28      hFontn := file_LoadImageControl
                ("NoName1.dan", "NoName1.gcn", 1);
```

The convention is to rename the variable “hFontx” declared on line 13 to “hFont1”.

```
13      var hFont1;   // Handle to
```

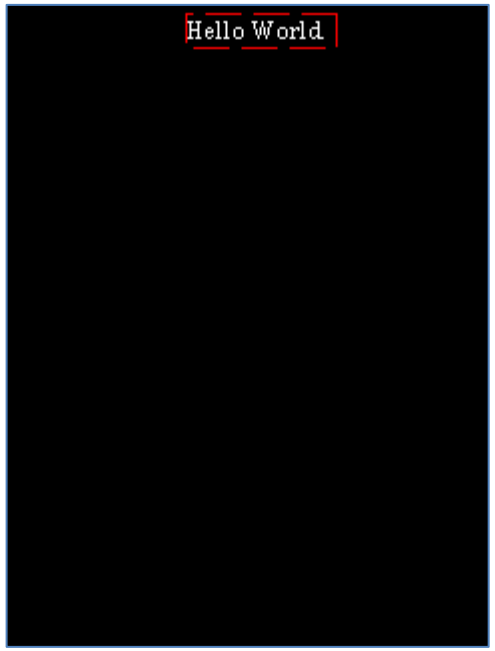
On line 28, replace the letter “n” with an integer, as shown below.

```
28      hFont1 := file_LoadImageControl
                ("NoName1.da1", "NoName1.gc1", 1); /
```

The variable “hFont1” is now the handle that will be used as a reference for the first font that will be added to the project.

Add a String Object

Starting from a blank WYSIWYG screen, add a string object which contains the string “Hello World” and set the font to “Cambria”. This is Strings1. The process for doing this has been discussed previously. Below is the result.



Object Inspector

Form	Form1
Object	Strings1

Properties | Paste Code

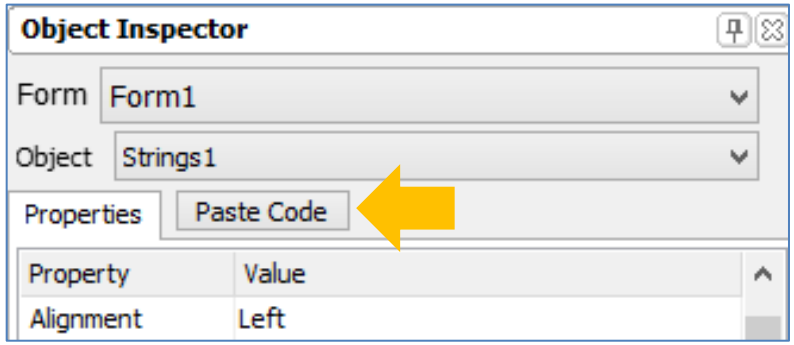
Property	Value
Alignment	Left
BGcolor	BLACK
Height	18
FGcolor	WHITE

Paste the Code for a String Object

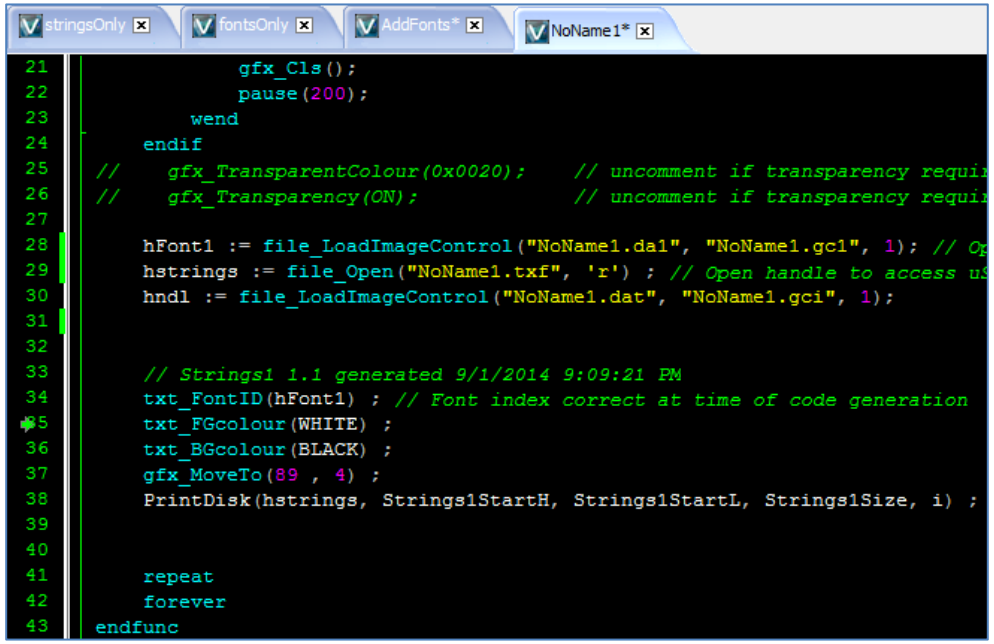
Put the cursor somewhere just before the indefinite repeat-forever loop as shown below.

```
17      if (!(disk:=file_Mount()))
18          while(!(disk :=file_Mount()))
19              putstr("Drive not mounted...");
20              pause(200);
21              gfx_Cls();
22              pause(200);
23          wend
24      endif
25  //    gfx_TransparentColour(0x0020);     // uncomment if
26  //    gfx_Transparency(ON);              // uncomment if
27
28      hFont1 := file_LoadImageControl("NoName1.da1", "NoNa
29      hstrings := file_Open("NoName1.txf", 'r') ; // Open
30      hndl := file_LoadImageControl("NoName1.dat", "NoName
31
32
33
34      repeat
35      forever
36  endfunc
```

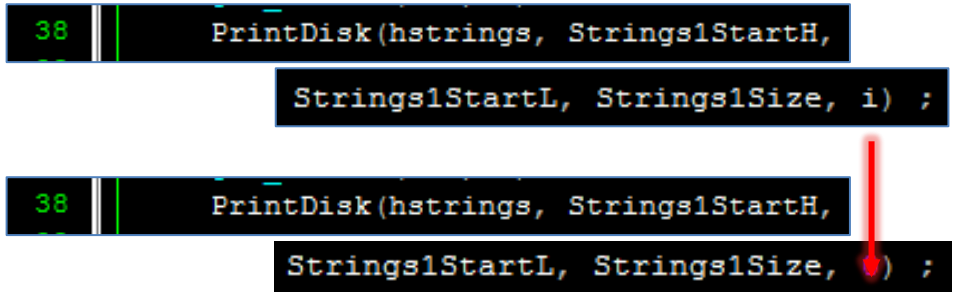
In the object inspector, click on the “Paste Code” button.



The code area is now updated.



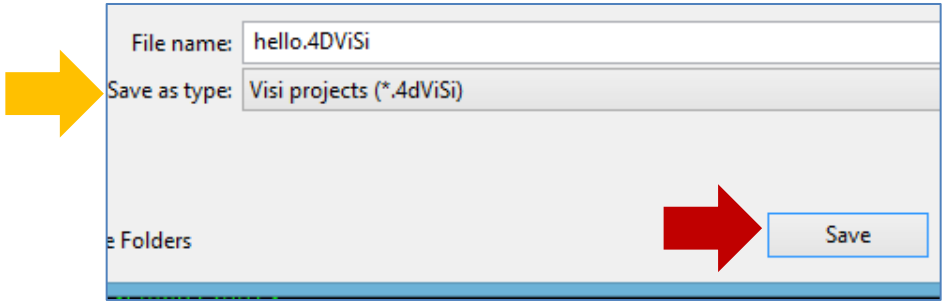
On line 38, change the fifth parameter of the function PrintDisk() from “i” to “0”.



Don’t forget to add the include file "**PrintDisk.inc**" to your 4DGL code.

```
#platform "uLCD-32PTU"

// Program Skeleton 1.1 generated 8/28/2014

#inherit "4DGL_16bitColours.fnc"

#inherit "VisualConst.inc"

#inherit "NoName1Const.inc"
#inherit "PrintDisk.inc"
func main()
```

Save the project with a name. Here it is saved as “Hello”.



Double check if the string and font files are renamed appropriately.

```
28      hFont1 := file_LoadImageControl
                ("hello.da1", "hello.gc1", 1);

29      hstrings := file_Open("hello.txf", 'r') ;
```

Attached is a project for demonstrating how to print a string from the uSD card using a font from the uSD card.

List of Attached Files

File	Description
stringsOnly.zip	Demonstrates how to print strings from the uSD card
fontsOnly.zip	Demonstrates how to use uSD card fonts
hello.zip	Demonstrates how to print a string from the uSD card using a font from the uSD card.
AddFonts.zip	Prints two string objects; uses two uSD card fonts.

Tips

- The changes you make on the object properties after pasting the code do not reflect on the code. That is if you wish to edit the object properties you need to “Paste Code” again after doing so.
- There is a complete STRINGSDEMO.4DViSi example in the 4D Workshop4 IDE. Click ‘Samples’, select Picaso ViSi – Click for filtered browse, look for,

  ..\4D Labs\Picaso Visi

Proprietary Information

Disclaimer of Warranties & Limitation of Liability