APPLICATION NOTE

# 4D SYSTEMS

*TURNING TECHNOLOGY INTO ART*

# ViSi Genie Writing to Genie Objects Using an Arduino Host

**Document Date: 23rd July 2014**

**Document Revision: 1.0**

# Description

This Application Note explores the possibilities provided by the ViSi-Genie environment in Workshop to work with an Arduino host. In this example, the host is an AVR ATmega328 microcontroller-based Arduino Uno board. The host can also be an Arduino Mega 2560 or Due. Ideally, the application described in this document should work with any Arduino board with at least one UART serial port. See specifications of Aduino boards here.

Before getting started, the following are required:

- Any of the following 4D Picaso display modules:

  uLCD-24PTU uLCD-28PTU uLCD-32PTU

  uLCD-32WPTU uLCD-43(P/PT/PCT) uVGA-III

  other superseded modules which support the ViSi Genie environment

- The target module can also be a Diablo16 display

  uLCD-35DT uLCD-70DT

  *See the section "Write to a Pin Output Object" when compiling this project for a Diablo16 display module.*

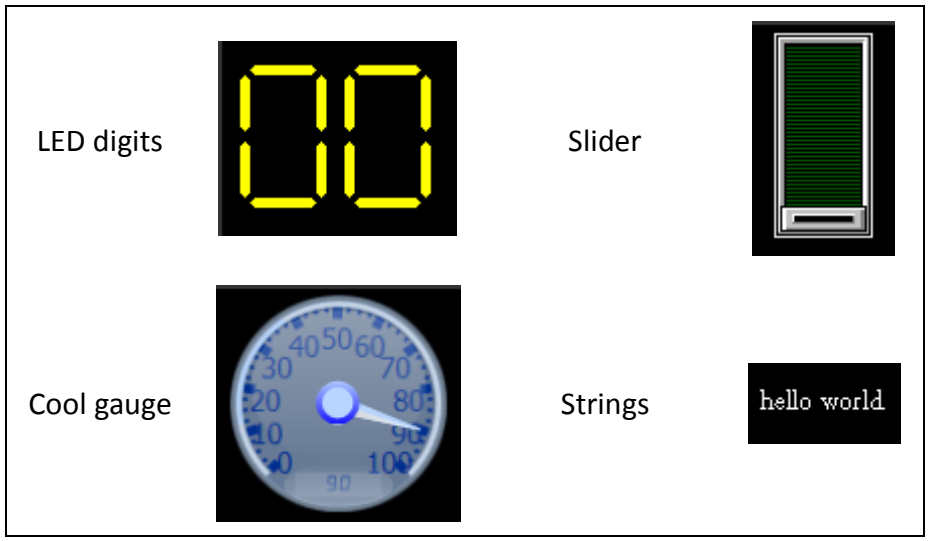
  Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor. The display module used in this application note is the uLCD-32PTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.

- 4D Programming Cable or µUSB-PA5
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- Any Arduino board with a UART serial port and Arduino IDE
- 4D Arduino Adaptor Shield (optional) or connecting wires
- 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

It is often difficult to design a graphical display without being able to see the immediate results of the application code. ViSi-Genie is the perfect software tool that allows users to see the instant results of their desired graphical layout with this large selection of gauges and meters (called widgets) that can simply be dragged and dropped onto the simulated module display. The following are some examples of widgets or objects used in this application note.




This document is a supplement to ViSi-Genie Connecting a 4D Display to an Arduino Host. The application of the Genie class member function **`WriteObject()`** function to different Genie objects is shown here. A ViSi-Genie program and an Arduino sketch are provided for demonstration purposes. The ViSi Genie program contains the different objects created in Workshop. The Arduino sketch contains the commands to control each of these objects. To learn how to create a ViSi Genie program, go to http://www.4dsystems.com.au/appnotes/. The page contains application notes which explain how to create and configure objects in a ViSi-Genie program.

# Setup Procedure

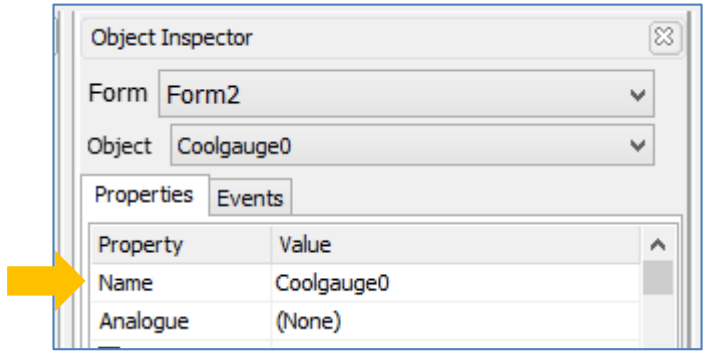
For instructions on how to launch Workshop 4, how to open a ViSi-Genie project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note:

ViSi Genie Getting Started – First Project for Picaso Displays (for Picaso)
or
ViSi Genie Getting Started – First Project for Diablo16 Displays (for Diablo16).

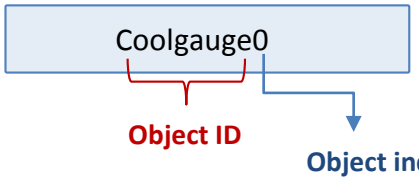
# Writing to Genie Objects Using an Arduino Host

## Naming of Objects

When creating objects in the Workshop IDE, objects are automatically named as they are created. For instance, the first cool gauge object added will be given the name Coolgauge0. The object name, along with the object properties, is shown by the Object Inspector.




Naming is important to differentiate between objects of the same kind. For example, suppose the user adds another cool gauge object to the WYSIWYG (What-You-See-Is-What-You-Get) screen. This object will be given the name Coolgauge1 – it being the second cool gauge in the program. The third cool gauge will be given the name Coolgauge2, and so on. An object's name therefore identifies its kind and its unique index number. It has an ID (or type) and an index.




It is important to take note of an object's ID and index. When programming in the Arduino IDE, an object's status can be polled or changed if its ID and index are known, as will be shown in the next section.

## Writing to an Object

The status of a ViSi-Genie object can be controlled or changed by a host controller with the appropriate message sent thru the serial port. The format of this message is defined in the ViSi Genie Communications Protocol which is discussed in the ViSi Genie User Reference Manual. To write to a ViSi-Genie object, the format is:

| 2.1.2 Command and Parameters Table | | | | | | |
|---|---|---|---|---|---|---|
| **Command** | **Code** | **Parameter 1** | **Parameter 2** | **Parameter 3** | **Parameter 4** | **Checksum** |
| WRITE_OBJ | **0x01** | Object ID | Object Index | Value (msb) | Value(lsb) | Checksum |

The ViSi-Genie-Arduino library implements this format as a function, the prototype for which is declared under the Genie class.

```
WriteObject (uint16_t object,
                uint16_t index, uint16_t data);
```

The first parameter must be an integer which specifies the object ID. The second parameter is an integer which specifies the index of the object. The third parameter is an integer which holds the data to be written to the object. Example:

```
genie.WriteObject(GENIE_OBJ_COOL_GAUGE, 0x00, gaugeVal);
```

**Note that the third parameter must be an integer. Floats and other data types cannot be passed as an argument to this function.** The Arduino platform provides functions for converting floats to integers. The user can refer to the Arduino website for further information.

## Writing to a String Object

String objects can display predefined text (created in the ViSi-Genie environment) or dynamic text received from the host. The process of making a string object display predefined text is discussed in the section **"Write to a Predefined Strings Object"**. For the host to write a dynamically created text to a string object, the format of the message is:

| 2.1.2 Command and Parameters Table | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Command** | **Code** | **Parameter 1** | **Parameter 2** | **Parameter 3** | **Parameter 4** | **Parameter N** | **Checksum** |
| WRITE_STR | **0x02** | String Index | String Length | String (1 byte chars) | | | Checksum |

The function prototype for this in the ViSi-Genie-Arduino library is:

```
WriteStr     (uint16_t index, char *string);
```

The first parameter is the index of the string. The second parameter is a pointer to a null-terminated character array. Examples:

```
#define        GENIE_VERSION    "GenieArduino 31-Mar-2014"
genie.WriteStr(0, GENIE_VERSION);
```

```
char myArray[6] = {'h', 'e', 'l','l','o','\0'};
genie.WriteStr(0, myArray);
```

```
char myArray2[] = "Welcome to 4D Systems!";
genie.WriteStr(0, myArray2);
```

**Note that for all three examples, the second argument for `genie.WriteStr()` is a pointer to a null-terminated character array. Section 2.1.3.3 Write String (ASCII) Message of the ViSi-Genie Reference Manual emphasizes this.**

**Note1:** The ASCII characters are 1 byte each.

**Note2:** The String should be null terminated.

**Note3:** Refer to the application notes for detailed information on

It is a common mistake for beginners to use integers, strings, floats, and other data types instead of a pointer to a null-terminated character array as the second argument passed to **`genie.WriteStr()`** . The Arduino platform provides functions for converting integers, strings, and floats to null-terminated character arrays. The user can refer to the Arduino website for further information.

The ViSi-Genie Arduino library handles the actual communication between the Arduino host and the display module, making sure that the message sent is of the correct format. This also includes error checksum coding, acknowledgment, etc. The user will just have to specify the strings object index and the string to be displayed when using `genie.WriteStr()`.

The user can easily associate the syntax of the functions to the corresponding format in the ViSi-Genie Communications Protocol. Table 3.3 of the manual shows the objects and their ID numbers.

| Object | ID | Gauge | 11 (0x0B) | Sound | 22 (0x16) |
|---|---|---|---|---|---|
| Dipswitch | 0 (0x00) | Image | 12 (0x0C) | Timer | 23 (0x17) |
| Knob | 1 (0x01) | Keyboard | 13 (0x0D) | Spectrum | 24 (0x18) |
| Rockerswitch | 2 (0x02) | | | Scope | 25 (0x19) |
| Rotaryswitch | 3 (0x03) | Led | 14 (0x0E) | Tank | 26 (0x1A) |
| Slider | 4 (0x04) | Leddigits | 15 (0x0F) | UserImages | 27 (0x1B) |
| Trackbar | 5 (0x05) | Meter | 16 (0x10) | PinOutput | 28 (0x1C) |
| Winbutton | 6 (0x06) | Strings | 17 (0x11) | PinInput | 29 (0x1D) |
| Angularmeter | 7 (0x07) | Thermometer | 18 (0x12) | 4Dbutton | 30 (0x1E) |
| Coolgauge | 8 (0x08) | Userled | 19 (0x13) | AniButton | 31 (0x1F) |
| Customdigits | 9 (0x09) | Video | 20 (0x14) | ColorPicker | 32 (0x20) |
| Form | 10 (0x0A) | Statictext | 21 (0x15) | UserButton | 33 (0x21) |

In the ViSi-Genie-Arduino library, the ID numbers are then used to define the Genie object constants.

```
GENIE_OBJ_DIPSW          0
GENIE_OBJ_KNOB           1
GENIE_OBJ_ROCKERSW       2
GENIE_OBJ_ROTARYSW       3
GENIE_OBJ_SLIDER         4
GENIE_OBJ_TRACKBAR       5
GENIE_OBJ_WINBUTTON      6
GENIE_OBJ_ANGULAR_METER  7
GENIE_OBJ_COOL_GAUGE     8
GENIE_OBJ_CUSTOM_DIGITS  9
GENIE_OBJ_FORM           10
GENIE_OBJ_GAUGE          11
GENIE_OBJ_IMAGE          12
GENIE_OBJ_KEYBOARD       13
GENIE_OBJ_LED            14
GENIE_OBJ_LED_DIGITS     15
GENIE_OBJ_METER          16
GENIE_OBJ_STRINGS        17
```

```
GENIE_OBJ_THERMOMETER    18
GENIE_OBJ_USER_LED       19
GENIE_OBJ_VIDEO          20
GENIE_OBJ_STATIC_TEXT    21
GENIE_OBJ_SOUND          22
GENIE_OBJ_TIMER          23
GENIE_OBJ_SPECTRUM       24
GENIE_OBJ_SCOPE          25
GENIE_OBJ_TANK           26
GENIE_OBJ_USERIMAGES     27
GENIE_OBJ_PINOUTPUT      28
GENIE_OBJ_PININPUT       29
GENIE_OBJ_4DBUTTON       30
GENIE_OBJ_ANIBUTTON      31
GENIE_OBJ_COLORPICKER    32
GENIE_OBJ_USERBUTTON     33
```

The following sections now show how the **genie.WriteObject()** function is applied to different Genie objects. Writing to the keyboard, static text, image, and pin input objects is not possible.

## Write to a DIP Switch

To write to a DIP switch:

```
genie.WriteObject(GENIE_OBJ_DIPSW, 0x00, 0);
delay(3000);
```

A three-second delay is added for the observer to see the object at the current state. Note that the function has three arguments as defined in the ViSi-Genie-Arduino library. The first argument is the Genie object to be written to, the second is the object index, and the third is the value which represents the state of the DIP switch. Thus, the command

```
genie.WriteObject(GENIE_OBJ_DIPSW, 0x00, 0);
delay(3000);
```

yields the result

State 0

The command

```
genie.WriteObject(GENIE_OBJ_DIPSW, 0x00, 1);
delay(3000);
```

yields the result

State 1

Table 2.1.2 shows the format for writing to objects.

2.1.2 Command and Parameters Table

| Command | Code | Parameter 1 | Parameter 2 | Parameter 3 | Parameter 4 | Checksum |
|---|---|---|---|---|---|---|
| WRITE_OBJ | 0x01 | Object ID | Object Index | Value (msb) | Value(lsb) | Checksum |

For further information, refer to the ViSi Genie User Reference Manual. The document gives an informative description of all of the Genie objects in relation to the ViSi Genie communications protocol.

## Write to a Knob

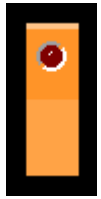
```
//write to a knob
genie.WriteStr(0, "Knob0 at \nvarious states");
for(i = 0; i<100; i++){
  genie.WriteObject(GENIE_OBJ_KNOB, 0x00, i);
  delay(100);
}
```

The code above will make Knob0 change its state from 0 to 99, hence making it appear to rotate.

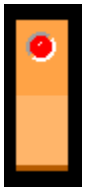
## Write to a Rocker Switch

```
//write to a rockerswitch
genie.WriteStr(0, "Rockerswitch0 at 0");
genie.WriteObject(GENIE_OBJ_ROCKERSW, 0x00, 0);
delay(3000);
genie.WriteStr(0, "Rockerswitch0 at 1");
genie.WriteObject(GENIE_OBJ_ROCKERSW, 0x00, 1);
delay(3000);
```

Similar to the DIP switch example, the code above displays the rocker switch at state 0 and then at state 1.



State 0



State 1

## Write to a Rotary Switch

```
//write to a rotary switch
genie.WriteStr(0, "Rotaryswitch0 at\n various states");
for(i = 0; i<9; i++){
  genie.WriteObject(GENIE_OBJ_ROTARYSW, 0x00, i);
  delay(350);
}
```

The code above will make Rotaryswitch0 change its state from 0 to 8. To learn how to configure a rotary switch, refer to ViSi-Genie Inputs.

## Write to a Slider

```
//write to a slider
genie.WriteStr(0, "Slider0 at \nvarious states");
for(i = 0; i<100; i++){
  genie.WriteObject(GENIE_OBJ_SLIDER, 0x00, i);
  delay(100);
}
```

Similar to the knob example, the code above will make Slider0 change its state from 0 to 99.

## Write to a Track Bar

```
//write to a trackbar
genie.WriteStr(0, "Trackbar0 at \nvarious states");
for(i = 0; i<100; i++){
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x00, i);
  delay(100);
}
```

The code above will make Trackbar0 change its state from 0 to 99.

## Write to a Winbutton

```
//write to a winbutton
genie.WriteStr(0, "Winbutton0 at 0");
genie.WriteObject(GENIE_OBJ_WINBUTTON, 0x00, 0);
delay(3000);
genie.WriteStr(0, "Winbutton0 at 1");
genie.WriteObject(GENIE_OBJ_WINBUTTON, 0x00, 1);
delay(3000);
```

Similar to the DIP switch example, the code will show Winbutton0 at state 0 and then at state 1.


State 0


State 1

The button used in this example is configured as a toggle button. ViSi-Genie Advanced Buttons explains how to create a toggle button.

## Navigate to a New Form

```
//navigate to Form1
genie.WriteObject(GENIE_OBJ_FORM, 0x01,0);
genie.WriteStr(1, "This is now \nForm1");
```

The second parameter, 0x01 is the index of the form to be activated. The third argument can be of any value since the Genie communications protocol does not require a value (MSB and LSB). See section 3.2.6.1 of the ViSi Genie User Reference Manual.

## Write to an Angular Meter

```
//write to an angular meter
genie.WriteStr(1, "Angularmeter0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_ANGULAR_METER, 0x00, i);
delay(100);}
```

The code above will make Angularmeter0 change its state from 0 to 99.

## Write to a Cool Gauge

```
//write to a cool gauge
genie.WriteStr(2, "Coolgauge0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_COOL_GAUGE, 0x00, i);
delay(100);}
```

The code above will make Coolgauge0 change its state from 0 to 99.

## Write to a Custom Digits

```
//write to a custom digit
genie.WriteStr(3, "Customdigits0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x00, i);
delay(100);}
```

The code above will make Customdigits0 change its state from 0 to 99. To learn how to create a custom digits object, open the ViSi sample program in Workshop under File menu – Samples – Picaso ViSi – CLOCK. The block comment discusses how the bitmap image of the digits was created.

## Write to a Gauge

```
//write to a gauge
genie.WriteStr(3, "Gauge0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_GAUGE, 0x00, i);
delay(100);}
```

The code above will make Gauge0 change its state from 0 to 99.

## Write to an LED

```
//write to an LED
genie.WriteStr(3, "Led0 at 0");
genie.WriteObject(GENIE_OBJ_LED, 0x00, 0);
delay(3000);
genie.WriteStr(3, "Led0 at 1");
genie.WriteObject(GENIE_OBJ_LED, 0x00, 1);
delay(3000);
```

Similar to the DIP switch example, the code above displays the LED at state 0 and then at state 1.


State 0


State 1

## Write to a LED Digits

```
//write to a LED digits
genie.WriteStr(3, "Leddigits0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, i);
delay(100);}
```

The code above will make Leddigits0 change its state from 0 to 99.

## Write to a Thermometer

```
//write to a thermometer
genie.WriteStr(4, "Thermometer0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_THERMOMETER, 0x00, i);
delay(100);}
```

The code above will make Thermometer0 change its state from 50 to 149. Note that the value sent by the Arduino host is offset by 50. The user has to account for this offset.

## Write to a Meter

```
//write to a meter
genie.WriteStr(4, "Meter0 at \nvarious states");
for(i = 0; i<100; i++){
genie.WriteObject(GENIE_OBJ_METER, 0x00, i);
delay(100);}
```

The code above will make Meter0 change its state from 0 to 99.

## Write to a User LED

```
//write to a user LED
genie.WriteStr(5, "Userled0 at 0");
genie.WriteObject(GENIE_OBJ_USER_LED, 0x00, 0);
delay(3000);
genie.WriteStr(5, "Userled0 at 1");
genie.WriteObject(GENIE_OBJ_USER_LED, 0x00, 1);
delay(3000);
```

The code above displays the Userled0 at state 0 and then at state 1.

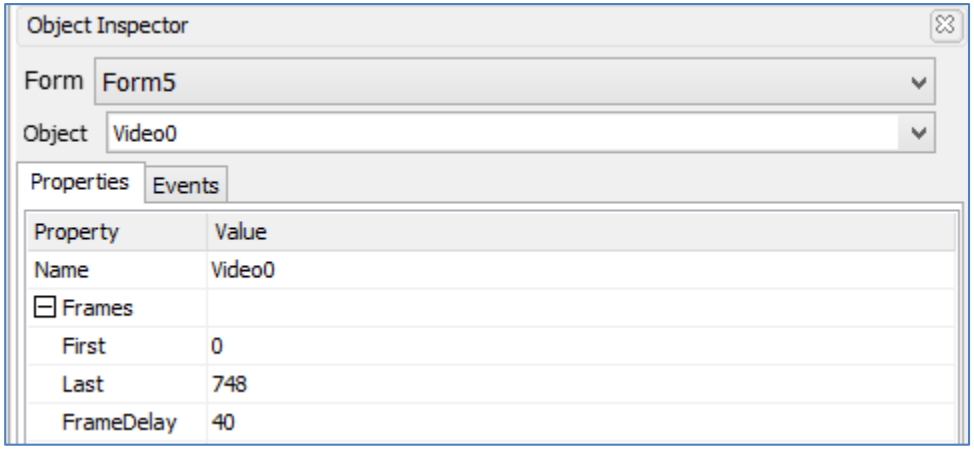
State 0

State 1

## Write to a Video

```
//write to a video
genie.WriteStr(5, "display the different \nframes of Video0");
for(i = 0; i<749; i++){
genie.WriteObject(GENIE_OBJ_VIDEO, 0x00, i);
delay(40);}
```

The code above will play Video0. Take note of the frame values and delay. The Object Inspector shows the frame properties of a video.




The FrameDelay (milliseconds) is equal to the reciprocal of the frame rate (fps).

## Write to a Predefined Strings Object

Besides displaying a dynamically created ASCII text received from the Arduino host, the user also has the option of displaying a predefined strings object created in the Workshop IDE. The document ViSi-Genie Labels, Text, and Strings discusses how predefined strings objects are created. Using predefined values makes the most efficient use of the communication link and also minimizes the code required in the host controller. In the ViSi Genie sample program, Strings6 contains four pages of text. Each of this page can be displayed by using the genie.WriteObject() function.

```
//write to a predefined strings object
genie.WriteStr(5, "Displaying a \npredefined\nstrings object
delay(5000);
genie.WriteStr(5, "page 1 of Strings6\nis intentionally\nlef
genie.WriteObject(GENIE_OBJ_STRINGS, 0x06, 0);
delay(3000);
genie.WriteStr(5, "page 2 of Strings6");
genie.WriteObject(GENIE_OBJ_STRINGS, 0x06, 1);
delay(7000);
genie.WriteStr(5, "page 3 of Strings6");
genie.WriteObject(GENIE_OBJ_STRINGS, 0x06, 2);
delay(7000);
genie.WriteStr(5, "page 4 of Strings6");
genie.WriteObject(GENIE_OBJ_STRINGS, 0x06, 3);
delay(7000);
```

Note that genie.WriteStr( ) is for displaying a dynamically created string (from the host), while genie.WriteObject( ) is for displaying a predefined string (stored in the uSD card).

## Write to a Timer

A timer object created in the Workshop IDE can be started or stopped by the host controller using the appropriate commands. Form0 of the ViSi Genie program has a timer object, Timer0, linked to Video1. When Timer0 starts, Video1 plays. Note that timer and sound objects always reside in Form0.

```
//write to a timer
genie.WriteStr(7, "Start Timer0");
genie.WriteObject(GENIE_OBJ_TIMER, 0x00,1);
delay(5000);
genie.WriteStr(7, "Stop Timer0");
genie.WriteObject(GENIE_OBJ_TIMER, 0x00,0);
delay(5000);
genie.WriteStr(7, "Timer0 resumes");
genie.WriteObject(GENIE_OBJ_TIMER, 0x00,1);
delay(5000);
genie.WriteStr(7, "Timer0 stops");
genie.WriteObject(GENIE_OBJ_TIMER, 0x00,0);
genie.WriteObject(GENIE_OBJ_VIDEO, 0x01,0); //rese
delay(5000);
```

## Write to a Sounds Object

The sounds object is a special object such that there can only be one instance of it in a Genie program. Similar to the timer object, the sounds object is invisible and always resides in Form0. The document ViSi-Genie Play Sound explains how to create and control a sounds object. Section 3.2.6.4 of the ViSi Genie User Reference Manual explains how to control a sounds object when using a host controller. The code below shows how this is done when programming an Arduino host. The ViSi Genie program for this code has a Sounds object containing three tracks.

```
//start playing track 1
genie.WriteStr(8, "Play track 1");
genie.WriteObject(GENIE_OBJ_SOUND, 0x00,0);
//set volume
genie.WriteStr(9, "volume = 100");
genie.WriteObject(GENIE_OBJ_SOUND, 0x01,100);
delay(7000);

//start playing track 2
genie.WriteStr(8, "Play track 2");
genie.WriteObject(GENIE_OBJ_SOUND, 0x00,1);
//control volume
genie.WriteStr(9, "volume = 75");
genie.WriteObject(GENIE_OBJ_SOUND, 0x01,75);
delay(7000);
//pause current track (track 2)
genie.WriteStr(8, "Pause track 2");
genie.WriteObject(GENIE_OBJ_SOUND, 0x02,0);//t
delay(3000);
//resume current track (track 2)
genie.WriteStr(8, "Continue \ntrack 2");
genie.WriteObject(GENIE_OBJ_SOUND, 0x03,0);//t
delay(5000);
//stop current track (track 2). Track goes bac
genie.WriteStr(8, "Stop track 2");
genie.WriteObject(GENIE_OBJ_SOUND, 0x04,0);//t
delay(3000);
```

```
//start playing track 3
genie.WriteStr(8, "Play track 3");
genie.WriteObject(GENIE_OBJ_SOUND, 0x00,2);
genie.WriteStr(9, "volume = 100");
genie.WriteObject(GENIE_OBJ_SOUND, 0x01,100);
delay(7000);
//stop current track (track 3). Track goes ba
genie.WriteStr(8, "Stop track 3");
genie.WriteObject(GENIE_OBJ_SOUND, 0x04,0);//
delay(3000);
```

The user is encouraged to open the accompanying Arduino sketch file and read the comments.

## Write to a Spectrum Object

The spectrum object is described with more detail in ViSi-Genie Spectrum.

```
for(i = 0; i < 250; i++){
  bar = random(0, 23);
  value = random(0,99);
  combined = (bar << 8) | value;
  genie.WriteObject(GENIE_OBJ_SPECTRUM, 0x00, combined);
  delay(20);
}
```

## Write to a Scope Object

The scope object is described in ViSi-Genie Single Trace Scope.

```
for(i = 0; i < 263; i++){
  genie.WriteObject(GENIE_OBJ_SCOPE, 0x00, random(0, 160));
  delay(50);
}
```

## Write to a Tank Object

The tank object is documented in ViSi-Genie Tank.

```
for(i = 0; i < 100; i++){
  genie.WriteObject(GENIE_OBJ_TANK, 0x00, i);
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x01, i);
  delay(50);
}
```

## Write to a User Images Object

The user images object is documented in ViSi-Genie User Images.

```
for(i = 0; i < 10; i++){
  j = random(0, 3);
  genie.WriteObject(GENIE_OBJ_USERIMAGES, 0x00, j);
  genie.WriteObject(GENIE_OBJ_USERIMAGES, 0x01, i);
  delay(1000);
}
```

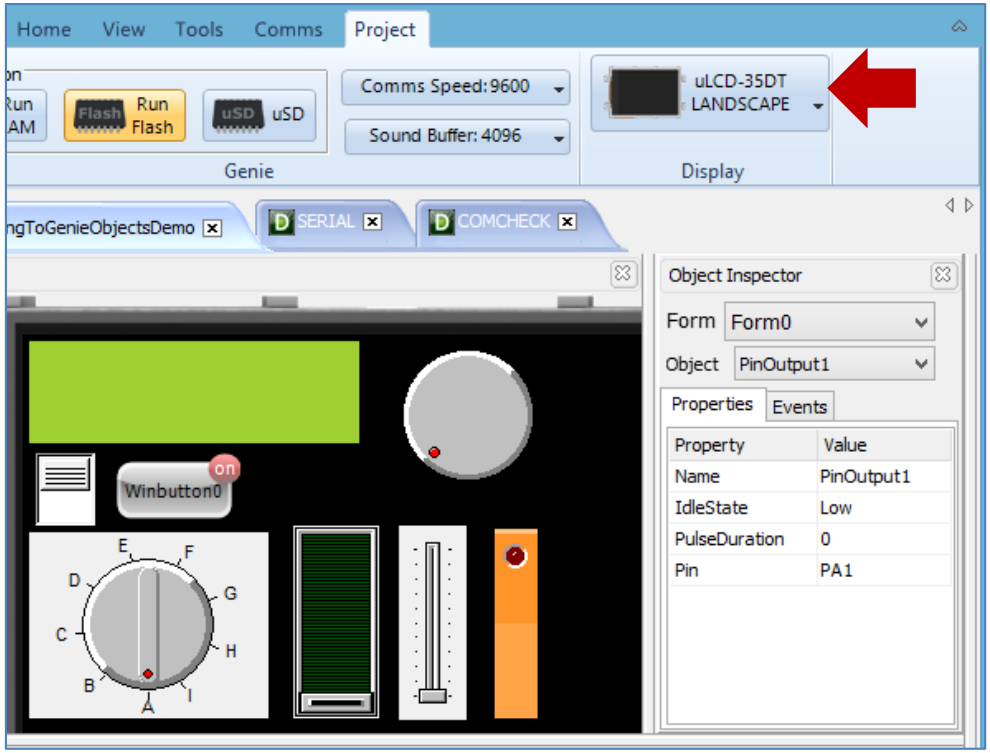
## Write to a Pin Output Object

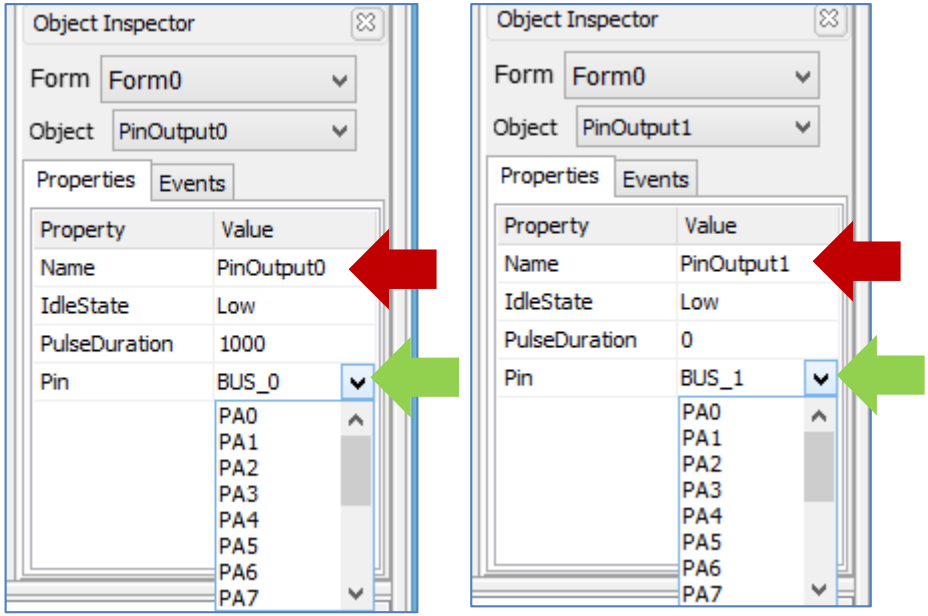
The pin output object is described in ViSi-Genie Pin Input and Output.

```
for(i = 0; i < 10; i++){
  genie.WriteObject(GENIE_OBJ_PINOUTPUT, 0x00, random(0,1));
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x01, 1);
  delay(1000);
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x01, 0);
  delay(1000);

  genie.WriteObject(GENIE_OBJ_PINOUTPUT, 0x01, j);
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, j);
  delay(50);
  j = !j;
}
```

Note that the pin labels of a Picaso display module are different from those of a Diablo16 display. Since the ViSi-Genie project attached to this application note was designed using a Picaso display, it has to be modified if recompiled for a Diablo16 display. Reconfigure the pin assignment of a pin output object by using the object inspector. For a uLCD-35DT target display for example, refer to the following images.






BUS_0 is a pin label for Picaso displays. Change the pin assignment of PinOutput0 by choosing a new pin label from the new list provided for Diablo16 displays. Do the same for PinOutput1. Failure to perform these steps will result to a compilation error. Refer to the datasheet of your display for more information.

## Write to a 4D Button Object

The 4D button object is described in ViSi-Genie 4D Buttons.

```
for(j = 0; j < 4; j++){
  genie.WriteObject(GENIE_OBJ_4DBUTTON, i, 1);
  delay(100);
  genie.WriteObject(GENIE_OBJ_4DBUTTON, i, 0);
  delay(100);
}
```

## Write to an Animated Button Object

The animated button object is described in ViSi-Genie Animated Button.

```
for(j = 0; j < 5; j++){
  genie.WriteObject(GENIE_OBJ_ANIBUTTON, 0x00, j);
  delay(100);
}
```

## Write to a Colour Picker Object

The colour picker object is described in ViSi-Genie Color Picker.

```
for(j = 0; j < 10; j++){
  i = random(0, 0xFFFF);
  genie.WriteObject(GENIE_OBJ_COLORPICKER, 0x00, i);
  delay(500);
}
```

## Write to a User Button Object

The colour picker object is described in ViSi-Genie User Button.

```
for(j = 0; j < 10; j++){
  genie.WriteObject(GENIE_OBJ_USERBUTTON, 0x00, j & 0x01);
  genie.WriteObject(GENIE_OBJ_USERBUTTON, 0x01, j & 0x01);
  delay(500);
}
```

# Proprietary Information



# 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.