APPLICATION NOTE

4D SYSTEMS

TURNING TECHNOLOGY INTO ART

Application Note: 4D-AN-P4018

ViSi Genie – Writing to Genie Objects Using an Arduino Host

Document Date: 25th July 2013

Document Revision: 1.0

Description

This Application Note explores the possibilities provided by ViSi-Genie 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 applications described in this document should work with any Arduino board with at least one UART serial port. See specifications of Aduino boards here.

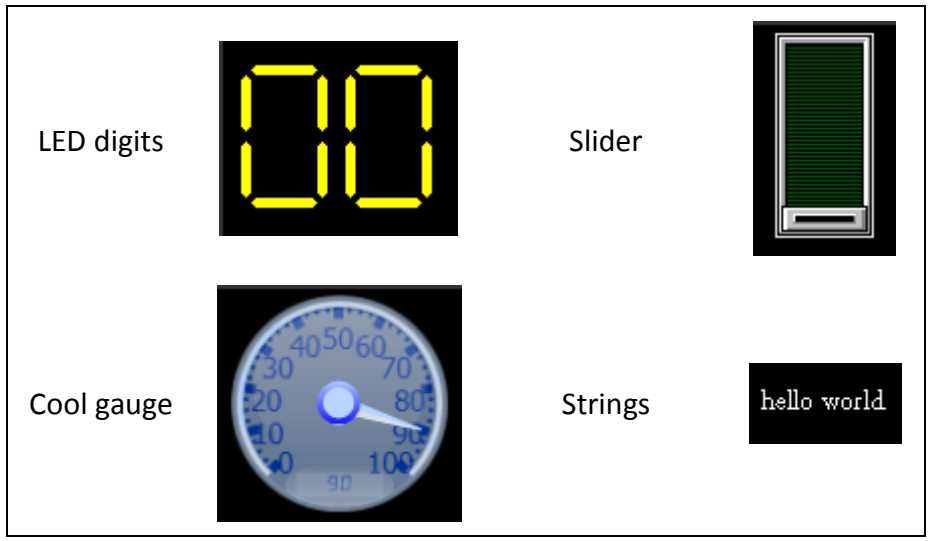
Before getting started, the following are required:

- 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  other superseded modules which support the ViSi Genie environment
- 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
- 4D Arduino Adaptor Shield (optional) or connecting wires
- Arduino IDE
- The user has completed the project described in 4D-AN-P4017-ViSi-Genie Connecting a 4D Display to an Arduino Host or has an understanding of how an Arduino host communicates with a 4D display

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 4D-AN-P4017-ViSi-Genie Connecting a 4D Display to an Arduino Host. The application of the **`genieWriteObject()`** function to different Genie objects is discussed here. A ViSi Genie program and an Arduino sketch are provided as demos. 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. ViSi-Genie – related app notes have names that start with 4D-AN-P400x.

Launch Workshop 4

There is a shortcut for Workshop 4 on the desktop.

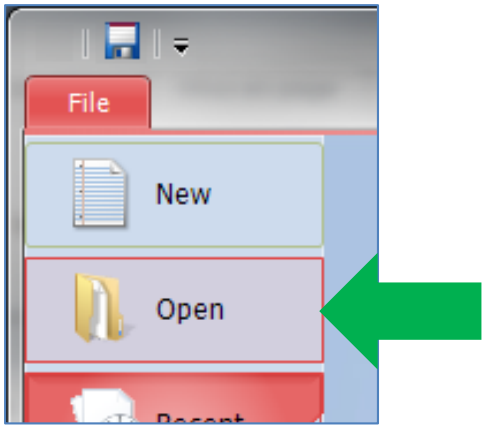
Launch Workshop 4 by double-clicking on the icon.

Setup Procedure

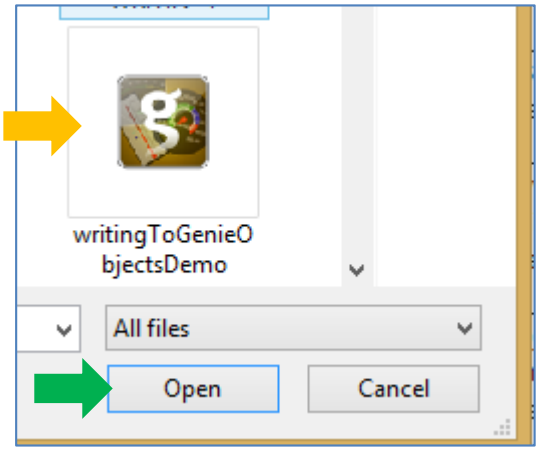
Load the Example

Workshop 4 opens and displays the **Recent** page.

To load the existing project, click on Open.



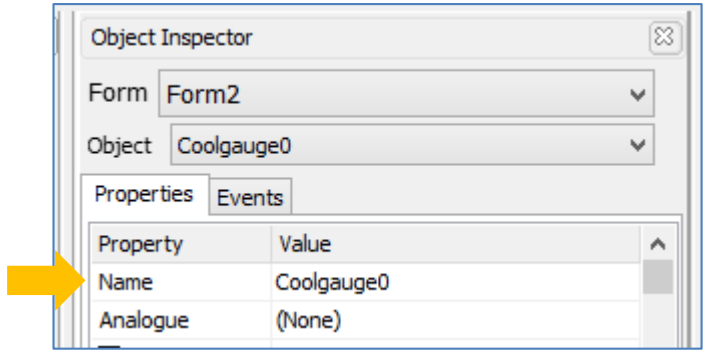
A standard open window asks for a ViSi-Genie project.



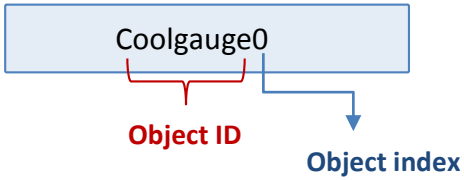
Writing to Genie Objects Using an Arduino Host

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.

Controlling the Status of an Object

The status of an object can be controlled or changed by a host controller by sending a message to the display. The format of this message is defined in the ViSi Genie Communications Protocol which is discussed in the ViSi Genie Reference Manual. In the ViSi-Genie-Arduino library, there are predefined functions for sending messages to the display, two of which are:

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

and

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

The functions are written such that the user can easily associate the syntax to the format specified in the ViSi-Genie Communications Protocol. Table 2.1.2 is posted below.

2.1.2 Command and Parameters Table

Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
READ_OBJ	0x00	Object ID	Object Index	-	-	-	Checksum
WRITE_OBJ	0x01	Object ID	Object Index	Value (msb)	Value(lsb)	-	Checksum
WRITE_STR	0x02	String Index	String Length	String (1 byte chars)			Checksum
WRITE_ STRU	0x03	String Index	String Length	String (2 byte chars)			Checksum
WRITE_ CONTRAST	0x04	Value	-	-	-	-	Checksum
REPORT_OBJ	0x05	Object ID	Object Index	Value (msb)	Value(lsb)	-	Checksum
REPORT_EVENT	0x07	Object ID	Object Index	Value (msb)	Value(lsb)	-	Checksum

This table can be found on page 6 of the ViSi Genie Reference Manual . It is highly recommended that the user downloads a copy of the manual. It will be used as a basis in determining the parameters or arguments of the `genieWriteObject()` and `genieWriteStr()` functions of the ViSi-Genie-Arduino library.

Table 3.3 on page 42 of the manual shows the objects and their ID numbers.

3.3. Object Summary Table

Object	ID	Input	Output	Notes
Dipswitch	0 (0x00)	✓	✓	
Knob	1 (0x01)	✓	✓	
Rockerswitch	2 (0x02)	✓	✓	
Rotaryswitch	3 (0x03)	✓	✓	
Slider	4 (0x04)	✓	✓	
Trackbar	5 (0x05)	✓	✓	
Winbutton	6 (0x06)	✓	✓	
Angularmeter	7 (0x07)		✓	
Coolgauge	8 (0x08)		✓	
Customdigits	9 (0x09)		✓	
Form	10 (0x0A)		✓	Used to set the current form
Gauge	11 (0x0B)		✓	
Image	12 (0x0C)			Displayed as part of form, no method to alter
Keyboard	13 (0x0D)	✓		Keyboard inputs are always single bytes and are unsolicited
Led	14 (0x0E)		✓	
Leddigits	15 (0x0F)		✓	
Meter	16 (0x10)		✓	
Strings	17 (0x11)		✓	
Thermometer	18 (0x12)		✓	
Userled	19 (0x13)		✓	
Video	20 (0x14)		✓	
Statictext	21 (0x15)			Displayed as part of form, no method to alter
Sound	22 (0x16)		✓	
Timer	23 (0x17)		✓	

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_KEYBOARD       13
GENIE_OBJ_ROCKERSW       2      GENIE_OBJ_LED            14
GENIE_OBJ_ROTARYSW       3      GENIE_OBJ_LED_DIGITS     15
GENIE_OBJ_SLIDER         4      GENIE_OBJ_METER          16
GENIE_OBJ_TRACKBAR       5      GENIE_OBJ_STRINGS        17
GENIE_OBJ_WINBUTTON      6      GENIE_OBJ_THERMOMETER    18
GENIE_OBJ_ANGULAR_METER  7      GENIE_OBJ_USER_LED       19
GENIE_OBJ_COOL_GAUGE     8      GENIE_OBJ_VIDEO          20
GENIE_OBJ_CUSTOM_DIGITS  9      GENIE_OBJ_STATIC_TEXT    21
GENIE_OBJ_FORM           10     GENIE_OBJ_SOUND          22
GENIE_OBJ_GAUGE          11     GENIE_OBJ_TIMER          23
GENIE_OBJ_IMAGE          12
```

The following sections now show how the `genieWriteObject()` and `genieWriteStr()` functions are applied to different Genie objects. Writing to the keyboard, static text, and image objects is not possible.

Send an ASCII Text to a Strings Object

In the main loop of the Arduino sketch, it is first shown how to send a dynamically created ASCII text to a strings object. The maximum strings length is 80. CRs and LFs can be included and the user is responsible for the 'formatting' of the string.

```
//write to a dipswitch and a text string
genieWriteStr(0, "Dipswitch0 at 0");
```

The text will be displayed by the object Strings0 on the screen. Note that two arguments are required – the first being the index of the strings object to be written to and the second being the text to be displayed. Compare the syntax of the command to the format defined in Table 2.1.2.

Command	Code	Parameter 1	Parameter 2	Parameter 3	Checksum
WRITE_STR	0x02	String Index	String Length	String (1 byte chars)	Checksum

The ViSi Genie Protocol specifies that the command WRITE_STR (used for sending ASCII text to the display) has a hexadecimal code of 0x02. Furthermore, there are three or more parameters – the strings object index, the length of the string, and the actual string characters. Lastly, the checksum is appended for error checking purposes. The reader is now directed to section 2.1.3.3 (page 10) of the ViSi Genie reference manual for more detailed 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 coding, acknowledgment, retransmission, etc. The user will just have to specify the strings object index and the string to be displayed when using `genieWriteStr()`.

Write to a DIP Switch

To write to a DIP switch:

```
genieWriteObject(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.

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

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

```
genieWriteObject(GENIE_OBJ_DIPSW, 0x00, 0);
delay(3000);
```

yields the result

State 0

The command

```
genieWriteObject(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 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

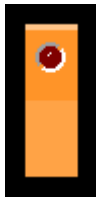
```
//write to a knob
genieWriteStr(0, "Knob0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(GENIE_OBJ_KNOB, 0x00, i);
delay(100);}
```

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

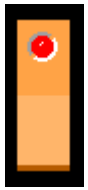
Write to a Rocker Switch

```
//write to a rockerswitch
genieWriteStr(0, "Rockerswitch0 at 0");
genieWriteObject(GENIE_OBJ_ROCKERSW, 0x00, 0);
delay(3000);
genieWriteStr(0, "Rockerswitch0 at 1");
genieWriteObject(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
genieWriteStr(0, "Rotaryswitch0 at\n various states");
for(i = 0; i<9; i++){
genieWriteObject(GENIE_OBJ_ROTARYSW, 0x00, i);
delay(350);}
```

The code above will make Rotaryswitch0 change its state from 0 to 8, making it appear to rotate. To learn how to configure a rotary switch, go to page 13 of 4D-AN-P4009 ViSi Genie Inputs.

Write to a Slider

```
//write to a slider
genieWriteStr(0, "Slider0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(0, "Trackbar0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(0, "Winbutton0 at 0");
genieWriteObject(GENIE_OBJ_WINBUTTON, 0x00, 0);
delay(3000);
genieWriteStr(0, "Winbutton0 at 1");
genieWriteObject(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. Page 17 of 4D-AN-P4004-ViSi-Genie-Advanced-Buttons explains how to create a toggle button.

Navigate to a New Form

```
//navigate to Form1
genieWriteObject(GENIE_OBJ_FORM, 0x01,0);
```

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 reference manual.

Write to an Angular Meter

```
//write to an angular meter
genieWriteStr(1, "Angularmeter0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(2, "Coolgauge0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(GENIE_OBJ_COOL_GAUGE, 0x00, i);
delay(100);}
delay(2000);
```

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

Write to a Custom Digits

```
//write to a custom digit
genieWriteStr(3, "Customdigits0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x00, i);
delay(100);}
delay(2000);
```

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
genieWriteStr(3, "Gauge0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(3, "Led0 at 0");
genieWriteObject(GENIE_OBJ_LED, 0x00, 0);
delay(3000);
genieWriteStr(3, "Led0 at 1");
genieWriteObject(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
genieWriteStr(3, "Leddigits0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(4, "Thermometer0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(4, "Meter0 at \nvarious states");
for(i = 0; i<100; i++){
genieWriteObject(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
genieWriteStr(5, "Userled0 at 0");
genieWriteObject(GENIE_OBJ_USER_LED, 0x00, 0);
delay(3000);
genieWriteStr(5, "Userled0 at 1");
genieWriteObject(GENIE_OBJ_USER_LED, 0x00, 1);
delay(3000);
```

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

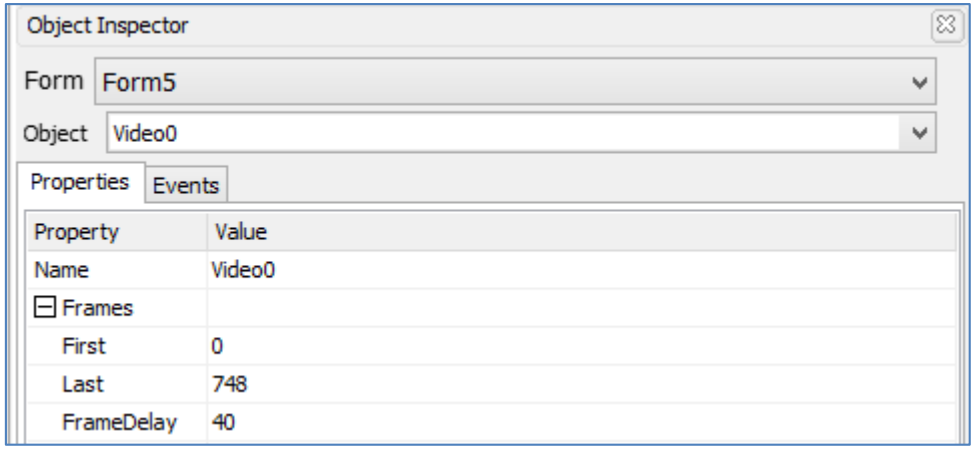
State 0

State 1

Write to a Video

```
//write to a video
genieWriteStr(5, "display the different \nframes of Video0");
for(i = 0; i<749; i++){
genieWriteObject(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 4D-AN-P4013-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 `genieWriteObject()` function.

```
//write to a predefined strings object
genieWriteStr(5, "page 1 of Strings6\nintentionally left\blank");
genieWriteObject(GENIE_OBJ_STRINGS, 0x06, 0);
delay(3000);
genieWriteStr(5, "page 2 of Strings6");
genieWriteObject(GENIE_OBJ_STRINGS, 0x06, 1);
delay(7000);
genieWriteStr(5, "page 3 of Strings6");
genieWriteObject(GENIE_OBJ_STRINGS, 0x06, 2);
delay(7000);
genieWriteStr(5, "page 4 of Strings6");
genieWriteObject(GENIE_OBJ_STRINGS, 0x06, 3);
delay(7000);
```

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. Form6 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
genieWriteStr(7, "Start Timer0");
genieWriteObject(GENIE_OBJ_TIMER, 0x00,1);
delay(5000);
genieWriteStr(7, "Stop Timer0");
genieWriteObject(GENIE_OBJ_TIMER, 0x00,0);
delay(5000);
genieWriteStr(7, "Timer0 resumes");
genieWriteObject(GENIE_OBJ_TIMER, 0x00,1);
delay(5000);
genieWriteStr(7, "Timer0 stops");
genieWriteObject(GENIE_OBJ_TIMER, 0x00,0);
genieWriteObject(GENIE_OBJ_VIDEO, 0x01,0); //reset Video1
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 4D-AN-P4006-ViSi-Genie-Play-Sound explains how to create and control a sounds object. Section 3.2.6.4 of the ViSi Genie 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.

```
//write to Sounds0. There can only be one sounds object.
//Object index is used to determine the command
//Object index      meaning (Value field)
//    0             Play wav file n
//    1             Set Volume n
//    2             Pause
//    3             Continue
//    4             Stop

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

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

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

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.