APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi-Genie: Arduino Update Diablo16 Program

DOCUMENT DATE:	9th MAY 2020
DOCUMENT REVISION:	1.0

WWW.4DSYSTEMS.COM.AU

Description

This application note shows how to update a ViSi-Genie program running in a Diablo16 display module.

Before getting started, the following are required:

Hardware

- Any 4D Systems display module powered by the Diablo16 processor
- Programming Adaptor for target display module
- 2 uSD Card or 1 uSD card for 4D Display and 1 SD card for Arduino
- USB Card Reader
- Arduino Mega with uSD/SD shield

Software

- Workshop4
- This requires the **PRO** version of Workshop4

This application note comes with two (2) ViSi-Genie projects as initial and updated projects.

Note: Using a non-4D programming interface could damage the processor and void the warranty.

Content

Description 2

Content 2

Application Overview 3

Setup Procedure 3

Create a New Project 3

Design the ViSi-Genie Projects 4

Opening Project Files 4

Magic Object Discussion 5

Design the Arduino Project 7

SD Card Access 7

Handling Update Request 7

Setting File Parameters 8

Sending the 4xe File 9

Handling Unsuccessful Attempts 10

Run the Program 10

Proprietary Information 11

Disclaimer of Warranties & Limitation of Liability 11

Application Overview

The Diablo16 processor has six flash banks (Bank 0 to Bank 5), each of which has a capacity of 32 kB. As of WS4 version 4.5.0.8, it is now possible for the user to specify the destination flash bank of a ViSi-Genie program. This was not possible in previous versions of Worskhop4. Prior to version 4.5.0.8, bank 0 was the only possible flash memory destination of a ViSi-Genie program.

The purpose of this application note is to show how to switch between banks using an Arduino as a host controller. This application note uses the ViSi-Genie environment, together with Genie-Magic. Thus, the PRO version of WS4 is required. Thus, the PRO version of WS4 is required. This application note is applicable to Diablo16 display modules only.

For more information on the basics of the multiple flash bank feature in ViSi-Genie, refer to the application note ViSi-Genie Flash Banks.

Setup Procedure

For instructions on how to launch Workshop4, 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 Diablo16 Display Modules

Create a New Project

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

- ViSi-Genie Getting Started - First Project for Diablo16 Display Modules

Design the ViSi-Genie Projects

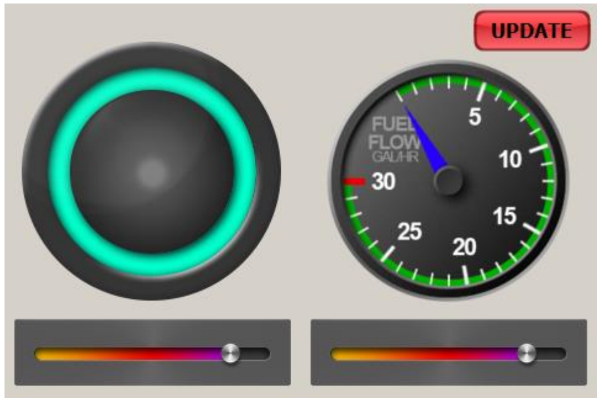
For this application note, a gen4-uLCD-35DCT-CLB will be used for the project. The same procedure is applicable for any Diablo16 displays. Also, this application note comes with zip files which contain demo projects needed for the discussions.

Opening Project Files

Open the project files inside the folders “Initial” and “Update”. It can be noticed that the projects have similar filenames. This is important since the filename of the graphics files (DAT and GCI files) is dependent on the project’s filename. This also denotes that the project is simply being updated.

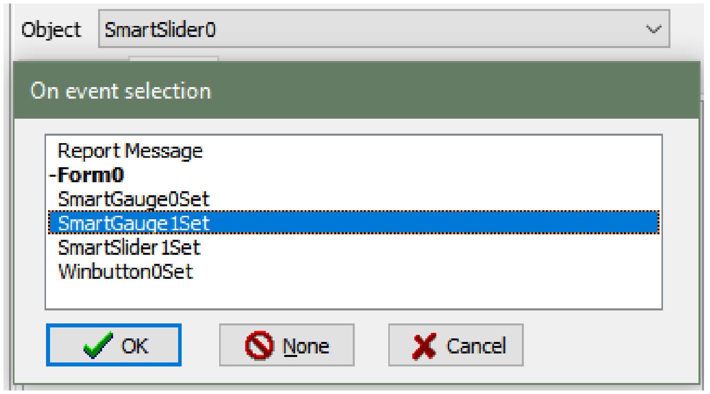
Note: This is only applicable when updating the program running without any changes to the graphics design.

Note that the projects contain magic objects and, so Workshop4 PRO is needed to open them. Both projects should contain the objects shown below.



The first version of the project is the one in the folder named “Initial”. This will be the initial program in flash bank 1 of the display.

This is purposely designed to act differently than the updated version. In this case, the Smart Slider on the left (SmartSlider0) has been set to update the Smart Gauge on the right (SmartGauge1)

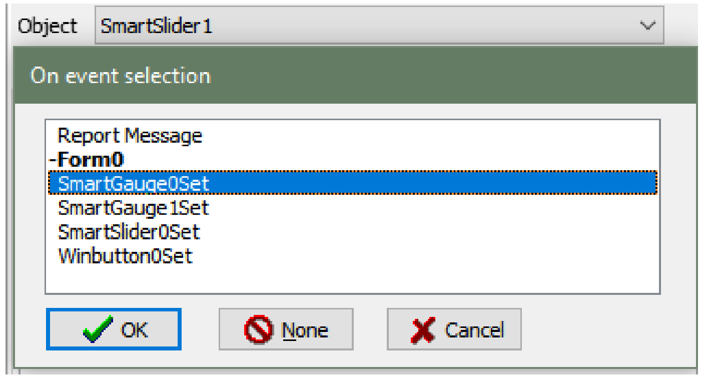


Object: SmartSlider0

On event selection

Report Message
-Form0
SmartGauge0Set
SmartGauge1Set
SmartSlider1Set
Winbutton0Set

while the Smart Slider on the right (SmartSlider1) has been set to update the Smart Gauge on the left (SmartGauge0).

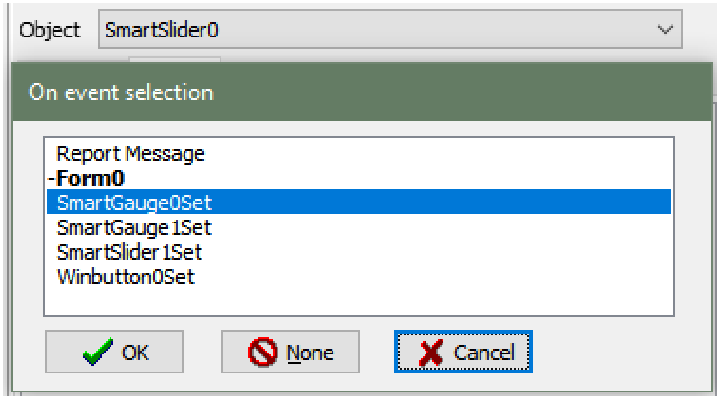


Object: SmartSlider1

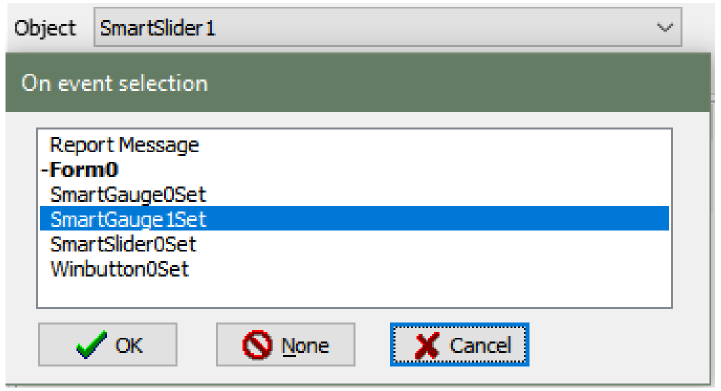
On event selection

Report Message
-Form0
SmartGauge0Set
SmartGauge1Set
SmartSlider0Set
Winbutton0Set

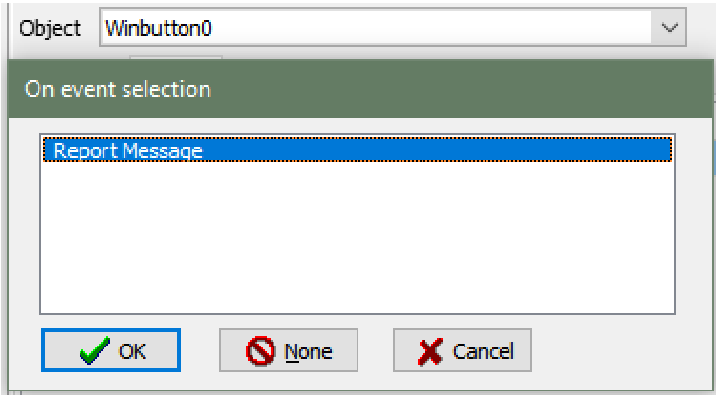
As you may have noticed, the first project seems to be either odd or wrong. So for the updated version which will be loaded to the Arduino's uSD, the Smart Slider on the left (SmartSlider0) will update the Smart Gauge on the left (SmartGauge0)



while the Smart Slider on the right (SmartSlider1) will update the Smart Gauge on the right (SmartGauge1).



Both projects also contain a Winbutton that simply reports an event to the host.



This Winbutton when pressed by user will tell the Arduino host that it wants an update.

Magic Object Discussion

A magic object is essential for this application. This object will be utilized for setting the current mode of the display's update routine, receiving information about the **4xe** file and receiving the file itself from the host.

```
#CONST
    // Display Replies
    ERR
    OK
    COMPLETE
#END
```

Shown are the replies sent by the display in response to the Arduino sending magic double bytes.

The display will reply ERR if the display had an error when writing the data into a temp file. Otherwise, it will send an OK or COMPLETE response. The COMPLETE response is only sent by the display if all the packets has been received successfully.

```
case WRITE_MAGIC_BYTES:
  if (newVal != 8)
    seroutX(NAK);
    break;
  endif
  var i, j;
  txt_MoveCursor(5, 0);
  for (i : = 0; i < 4; i++)          // change endianness
    j : = i * 2 ;
    fileData[i] : = (ptr[j + 1] << 8) + ptr[j] ;
  next
  file_Erase("temp0.4xe");
  seroutX(ACK);
  break;
```

The code shown is for receiving the properties of the files that are being sent by the Arduino to the Diablo16 display using magic bytes.

As shown, the display will send a NAK if the Arduino sends an incomplete or an excessive data. Otherwise, the data sent will be processed and stored to an array and an ACK will be sent back to the host.

```
case WRITE_MAGIC_DBYTES:
  txt_MoveCursor(0, 0);
  if (fileData[3] == -1)
    print("File Information hasn't been set");
    SendReport(REPORT_OBJ, tMagicObject, 0, ERR);
    break;
  endif

  var file, rem;
  file : = file_Open("temp0.4xe", 'a');
```

```
  if (!file_Error())
    fileSize += newVal * 2;
    rem : = fileData[0] - fileSize;

    file_Write(str_Ptr(ptr),newVal*2-((rem==-1)? 1: 0), file);

    print("Remaining Bytes: ",((rem==-1) ? 0 : rem), "     ");
    if (rem <= 0)
      file[FILE_DATE] : = fileData[1];
      file[FILE_TIME] : = fileData[2];
      file_Close(file);
      var buf[10];
      to (buf); print("RunBank", [DEC]fileData[3], ".4xe");
      if (file_Erase(str_Ptr(buf)))
        if (file_Rename("temp0.4xe", str_Ptr(buf)))
          fileData[0] : = 0;
          fileData[1] : = 0;
          fileData[2] : = 0;
          fileData[3] : = -1;
          print("\nFile Completed");
          SendReport(REPORT_OBJ, tMagicObject, 0, COMPLETE);
        endif
      endif
      return;
    endif
    SendReport(REPORT_OBJ, tMagicObject, 0, OK);
    file_Close(file);
    return;
  endif

  print("Error Opening File");
  SendReport(REPORT_OBJ, tMagicObject, 0, ERR);
  break;
```

The code presented handles the magic double bytes from the host. This is where the display accepts the data and saves it to a temporary file which is renamed upon completion.

This case may send an ERR, OK or COMPLETED to the host controller depending on the status.

Design the Arduino Project

Arduino Due was used for this project but this application note procedure should work similarly with other versions.

This application note uses the version of genieArduino library found here. Please refer to the Arduino website for their guide on how to install a library.

SD Card Access

The Arduino will need access to a memory storage where it can read the updated 4xe file which is loaded using ViSi Genie upload to uSD Card functionality. In this case, an SD card is used.

The updated 4xe file may be moved to a folder or stay in the root directory. In this application, it is moved to a folder named "Updates".

Handling Update Request

The Arduino shall wait for a request from the display which in this case is handled by a Winbutton.

```
else if (Event.reportObject.cmd == GENIE_REPORT_EVENT) {
  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON) {
    if (Event.reportObject.index == 0) {
      updateDisplay = true;
    }
  }
}
```

This is received by the event handler which is attached to the library using:

```
genie.AttachEventHandler(myGenieEventHandler);
```

As shown above, a variable *updateDisplay* is set as *true*. In the main loop of the sketch, this will be processed:

```
if (updateDisplay) {
  switch (update4xe("/Updates")) {
    case USD_FAILED:
    case GENIE_NACK:
    case GENIE_FAILED:
      Serial.println("Retrying...");
      break;
    case GENIE_COMPLETED:
      updateDisplay = false;
      Serial.println("Update Successful");
      break;
    default:
      break;
}
```

That is only if the display is online. The *updateDisplay* will be set to *false* once the update is finished.

If successful, the Arduino will reset the display using its reset pin.

```
void resetDisplay() {

  // Set Pin to Output (4D Arduino Adaptor V2 - Disp. Reset)
  pinMode(RESETLINE, OUTPUT);
  digitalWrite(RESETLINE, 1);  // Reset the Display via D4
  unsigned long startTime = millis();

  Serial.println("Resetting...");
  while (genie.online()) genie.DoEvents();

  // Execute another DoEvents to process Disconnect Flag
  genie.DoEvents();
  Serial.print("Disconnected after ");
  Serial.print(millis() - startTime);
  Serial.println("ms");

  digitalWrite(RESETLINE, 0);  // unReset the Display via D4

  Serial.println("\nWaiting for reconnect");
  genie.timeout(1250); // Return timeout to default
  genie.recover(500);

  while (!genie.online()) {
    genie.DoEvents(); // Wait until display is ready
  }

  genie.recover(50);
  Serial.println("\nDisplay is ready");
}
```

Notice that it is waiting until the display is flagged as disconnected before releasing the reset pin and checking if the display has reconnected.

Setting File Parameters

The update starts with telling the display about the 4xe file. This includes the file size, date and time. This should also include the flash bank that is being updated.

This application note will not discuss in detail how to retrieve those information from the file. However, the Arduino sketch included with this application note uses a function that copies the file size, date and time to an integer array.

The flash bank to which the file should be copied to is passed to the *update4xe* function.

```
int update4xe(char *directory, uint8_t bank = 1)
```

The directory that contains the update file should also be passed. Note that if the bank is not stated when using this function, it will simply default to bank 1.

The function will search for the appropriate RunbankN.4xe file. N represents the bank number which is from 1 to 5.

This is read afterwards by the function *getDetails*.

```
bool getDetails(char* filename, uint16_t details[3])
```

This returns *true* if the file is found and the parameters has been read. Otherwise, this will return *false*.

```
char *bankFile = "RunBankN.4xe";
uint16_t versionDetails[4];
bankFile[7] = bank + '0';
if (!getDetails(bankFile, versionDetails)) {
  return USD_FAILED;
}
versionDetails[3] = bank;
```

As shown above, the filename is set according to the *bank* as passed by the user to the function *update4xe*. It can also be noticed that it will return *USD_FAILED* if the details wasn't retrieved successfully. Otherwise, it continues and sets the value of the last index of array *versionDetails* to the bank set by the user.

Afterwards, it sends magic bytes to the display.

```
if(genie.WriteMagicBytes(0,(uint8_t*)versionDetails,8)!=1){
  Serial.println("Failed to set parameters");
  return GENIE_NACK;
}
```

As shown above, the function *update4xe* returns *GENIE_NACK* if no acknowledgement from the display has been received.

After successfully setting the parameters, the file is reopened and prepared for reading.

```
int packets = versionDetails[0] / MAX_SIZE;
int lastPacketSize = versionDetails[0] % MAX_SIZE;
if (!file.open(&ipFold, bankFile, O_READ)) {
  Serial.println("Failed to open file");
  return USD_FAILED;
}
Serial.println("File Ready to be Copied");
```

Sending the 4xe File

If setting the parameters is successful, packets of data will be sent to the display as magic double bytes until the copy completes.

```
for (int i = 0; i < packets; i++) {
  Serial.print("Packets Remaining: ");
  Serial.println(packets - i + ((lastPacketSize) ? 1 : 0));
  uint8_t data[MAX_SIZE] = {0};
  file.read(data, MAX_SIZE);
  do {
    magicReply = NO_REPLY;
    waitingForReply = true;
    uint8_t result = genie.WriteMagicDBytes(0, (uint16_t*)data,
255, GENIE_REPORT_OBJ); // Wait for GENIE_REPORT_OBJ reply
```

```
    if (result == -1) {
      Serial.println("Got timeout. Consider increasing timeout
using genie.timeout()");
      file.close();
      return GENIE_FAILED;
    }
    genie.DoEvents(); // Process GENIE_REPORT_OBJ message

    if (waitingForReply) { // If the GENIE_REPORT_OBJ is not from
MagicObject0
      Serial.println("Reply wasn't from MagicObject0.");
      Serial.println("Try stopping all ReadObject functions and
clearing the RX Buffer before executing update4xe function.");
      file.close();
      return GENIE_FAILED;
    }

    if (magicReply == ERR) {
      Serial.println("Display sent error message");
    }

  } while (magicReply == ERR);
}

if (lastPacketSize) {
  Serial.print("Packets Remaining: "); Serial.println(1);
  uint8_t data[lastPacketSize];
  memset(data, 0, lastPacketSize);
  file.read(data, lastPacketSize);
  do {
    magicReply = NO_REPLY;
    waitingForReply = true;

    uint8_t result = genie.WriteMagicDBytes(0, (uint16_t*)data,
(uint8_t)(lastPacketSize / 2 + lastPacketSize % 2),
GENIE_REPORT_OBJ); // Wait for GENIE_REPORT_OBJ reply

    if (result == -1) {
      Serial.println("Got timeout. Consider increasing timeout
using genie.timeout()");
      file.close();
      return GENIE_FAILED;
    }
    genie.DoEvents(); // Process GENIE_REPORT_OBJ message
    if (waitingForReply) { // If the GENIE_REPORT_OBJ is not from
MagicObject0
```

```
      Serial.println("Reply wasn't from MagicObject0.");
      Serial.println("Try stopping all ReadObject functions and
clearing the RX Buffer before executing update4xe function.");
      file.close();
      return GENIE_FAILED;
    }

    if (magicReply == ERR) {
      Serial.println("Display sent error message");
    }

  } while (magicReply == ERR);
}
file.close();
```

While doing so, the function may return *GENIE_FAILED* under certain conditions.

If everything has been successful until this point, the function will check if the last reply received says it has completed.

```
if (magicReply != COMPLETE) {
  Serial.println("Display didn't send complete message");
  return GENIE_FAILED;
}
```

If not, the function will return *GENIE_FAILED*. Otherwise, it will reset the display and return GENIE_COMPLETED as shown below.

```
resetDisplay();
return GENIE_COMPLETED;
```

Handling Unsuccessful Attempts

Errors during an update may exist. These errors can be from accessing the uSD card or an error reply from the display.

```
if (updateDisplay) {
  switch (update4xe("/Updates")) {
    case USD_FAILED:
    case GENIE_NACK:
    case GENIE_FAILED:
      Serial.println("Retrying...");
      break;
    case GENIE_COMPLETED:
      updateDisplay = false;
      Serial.println("Update Successful");
      break;
    default:
      break;
}
```

As shown above, if any error occur, it will simply retry continuously. Otherwise, if the update is successful, it will set the *updateDisplay* variable to *false*. This can easily be modified to meet user requirements.

Run the Program

For instructions on how to save a **ViSi-Genie** 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 .

- ViSi-Genie Getting Started - First Project for Diablo16 Display Modules

Proprietary Information

Disclaimer of Warranties & Limitation of Liability