



ViSi-Genie: Switching Banks

DOCUMENT DATE: **28th FEBRUARY 2020**
DOCUMENT REVISION: **1.2**



Description

This application note shows how to switch between banks of the Diablo16 processor.

Before getting started, the following are required:

Hardware

- Any [4D Systems display module](#) powered by a Diablo16 processor
- [Programming Adaptor for target display module](#)
- [uSD Card](#)
- USB Card Reader

Software

- [Workshop4](#)

This application note comes with one (1) ViSi-Genie project:

- flash1To5Demo.zip, etc.

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 Project.....	4
<i>Switching Between Bank 1 and Bank 2</i>	<i>4</i>
Run the Program from another Bank	5
Load the Programs to the Target Banks	6
Communicate with a Host	6
<i>Banks 1 to 5 Demo.....</i>	<i>10</i>
<i>Important Notes.....</i>	<i>11</i>
Connect the Display Module to the PC	11
Proprietary Information	12
Disclaimer of Warranties & Limitation of Liability	12

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 Workshop4. 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. This application note uses the ViSi-Genie environment, together with Genie-Magic. 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

First ViSi-Genie Project for Diablo16

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

First ViSi-Genie Project for Diablo16

Design the Project

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.

Switching Between Bank 1 and Bank 2

Open the project files inside the zip files “bankS1.zip” and “bankS2.zip”. The project “bankS1” should look like as shown below. The target bank for project “bankS1” is bank 1.



The project “bankS2”, on the other hand, is shown below. The target bank for this project is bank 2.



Run the Program from another Bank

The programs “bankS1” and “bankS2” have fancy button objects whose function is to run the program from the other bank. The 4DGL command for running the program of a bank is

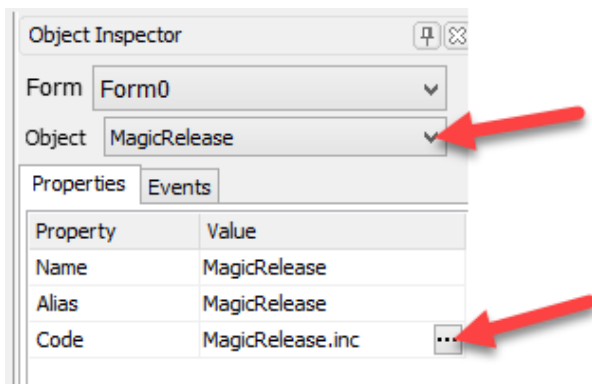
```
flash_Run(bank)
```

where **bank** is the flash bank to load the program from. Possible values of **bank** are:

```
0 or FLASHBANK_0
1 or FLASHBANK_1
2 or FLASHBANK_2
3 or FLASHBANK_3
4 or FLASHBANK_4
5 or FLASHBANK_5
```

For more information, refer to the [Diablo16 Internal Functions Manual](#).

This command is linked to the fancy buttons of the projects using a magic release object. Follow the procedure below to open the magic release object of project bankS1.



When Winbutton0 is released, `flash_Run(FLASHBANK_2)` is executed.

```

MagicRelease.inc
1 //
2 // Added 7/29/2017 8:37:26 AM
3 //
4 // Use 'ImageTouched' to detect 'current' object compar
5 // to determine the object for which touch has just bee
6 //
7
8 var returnFB := 0;
9
10 if(ImageTouched == iWinbutton0)
11     returnFB := flash_Run(FLASHBANK_2);
12 endif
13

```

Note that `flash_Run(FLASHBANK_2)` returns a value, which can be used to print error messages on the display.

```

MagicRelease.inc
9
10 if(ImageTouched == iWinbutton0)
11     returnFB := flash_Run(FLASHBANK_2);
12 endif
13
14 if(returnFB)
15     gfx_MoveTo(0,0);
16     if(returnFB == -1)
17         print("invalid bank number");
18     else if(returnFB == -2)
19         print("no valid program in the selected bank");
20     else
21         print("run bank not successful, unknown error");
22     endif
23     // ImageTouched := -1;
24     print("\nrestarting...");
25     pause(2000);
26     SystemReset();
27 endif

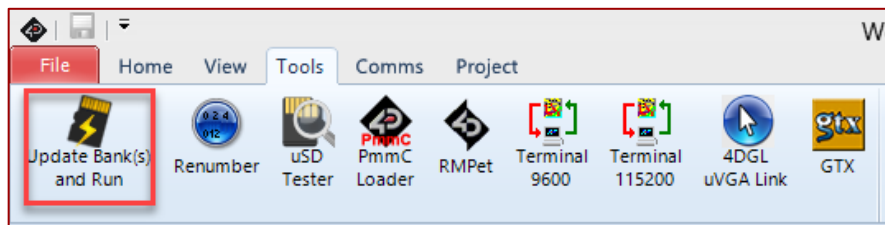
```

Load the Programs to the Target Banks

As discussed in the application note **ViSi-Genie Flash Banks**, there are two ways to load a program to its target bank. The first one is to directly load the program to the target flash bank. The second option is to copy the program file to the uSD card which is directly mounted to the PC. The uSD card can then be unmounted from the PC and mounted to the display module. Another program is then needed to copy the program from the uSD card to the destination flash bank. In this example, the second approach is used. It is also possible to use the first approach. For more information refer to the application note linked above.

Generate the program files and supporting files, and let WS4 copy them to the uSD card. Note that the projects contain magic objects, so WS4 PRO is needed to open them.

Unmount the uSD card from the PC and mount it to the display module. Load the Update Bank(s) and Run utility program to flash bank 0 of the display module.



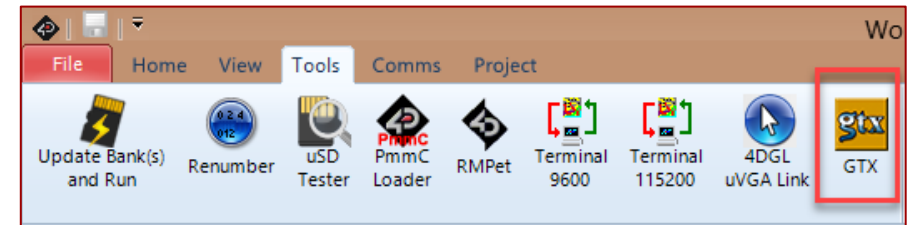
The utility program should now run and check the program files on the uSD card and copy them to their target bank if needed.

Communicate with a Host

When using an external host with the display, it might be useful for the program on the display to be able to do the following:

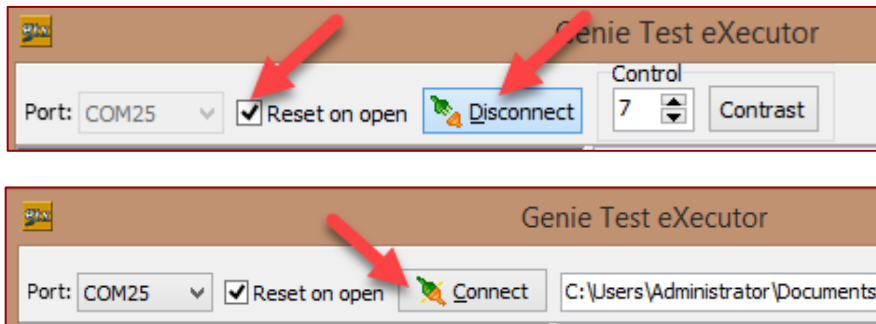
1. Send a message to the host containing information about the current bank number at the start
2. Receive and execute a command from the host for switching to another bank
3. Reply to a query from the host for the current flash bank number

These can be implemented using Genie Magic objects. Open the ViSi-Genie projects inside the zip folders “bankC1.zip” and “bankC2.zip”. The destination bank for “bankC1” is bank 1, “bankC2” is for bank 2. Update the display module with these programs then open the GTX tool under the Tools menu.

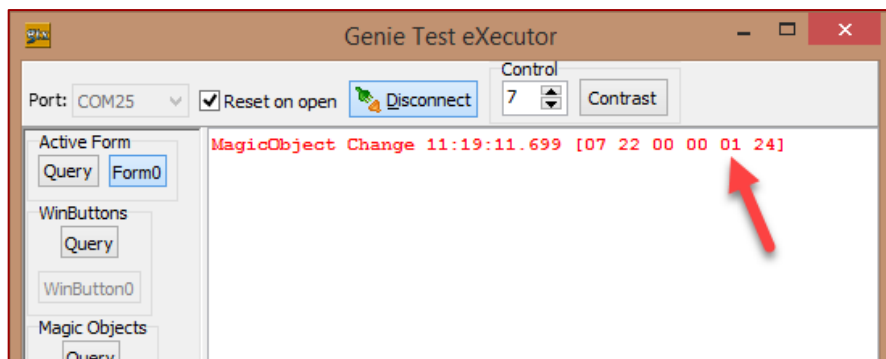


Make the Display Report the Current Bank to the Host

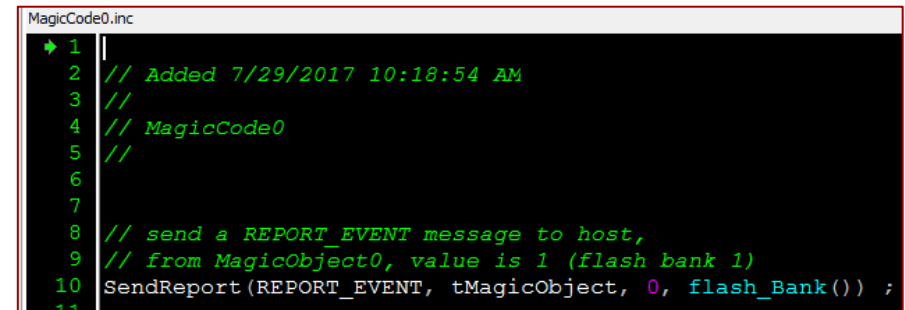
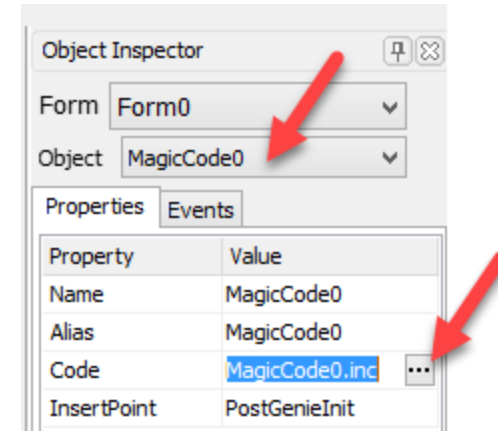
With the GTX tool open, enable the “Reset on open” option, click the Disconnect button, and click Connect.



This sequence of actions will reset the display module. The Update Bank(s) and Run program will then run, then control will be passed to bank 1. At the start of the program on bank 1, it will send a REPORT_EVENT message, the fifth byte of which is the number of the current bank.



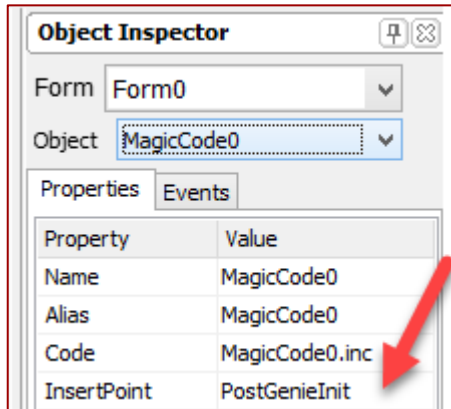
This is a result of the instruction on line 10 of MagicCode0 in bankC1 (and bankC2).



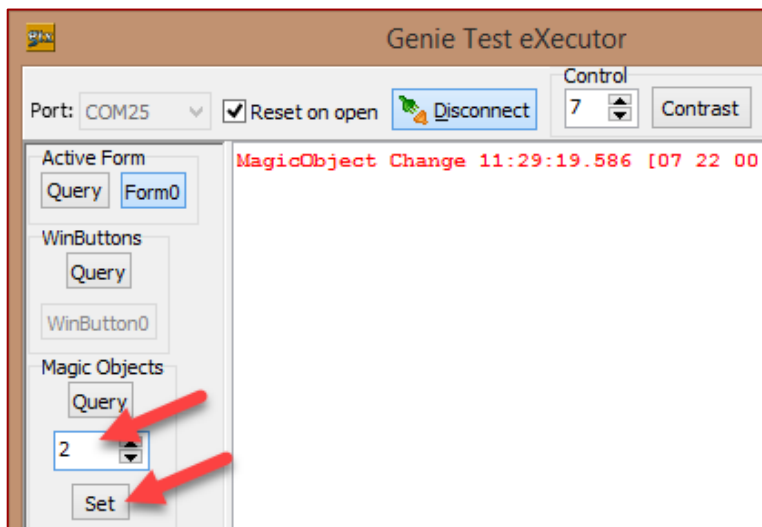
The 4DGL command `flash_Bank()` identifies which flash bank the code is running from. In this case, it will return 1 for flash bank 1.

Note further that MagicCode0 is inserted to PostGenieInit section of the ViSi-Genie program. For more information, refer to the [ViSi-Genie User Reference Manual](#).

Switch the Bank thru a WRITE_OBJ Message



In the GTX tool window, set the value of MagicObject0 to 2 and press set, as shown below.



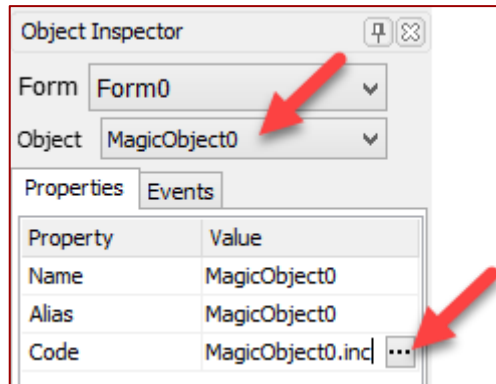
This sends a WRITE_OBJ message to the display module, with the value 2, which is the fifth byte indicated in the image below.

```
MagicObject Change 14:50:52.446 [07 22 00 00 01 24]
Set MagicObject Value 14:51:16.662 [01 22 00 00 02 21]
ACK 14:51:16.696 [06]
MagicObject Change 14:51:17.261 [07 22 00 00 02 27]
```

The display module will then interpret the fifth byte as the number of the new bank. The display module, at this point, should now have switched to bank 2. At the start of the program in bank 2, it will send a REPORT_EVENT message containing its bank number.

```
MagicObject Change 14:50:52.446 [07 22 00 00 01 24]
Set MagicObject Value 14:51:16.662 [01 22 00 00 02 21]
ACK 14:51:16.696 [06]
MagicObject Change 14:51:17.261 [07 22 00 00 02 27]
```


The WRITE_OBJ message coming from the host is processed by MagicObject0.



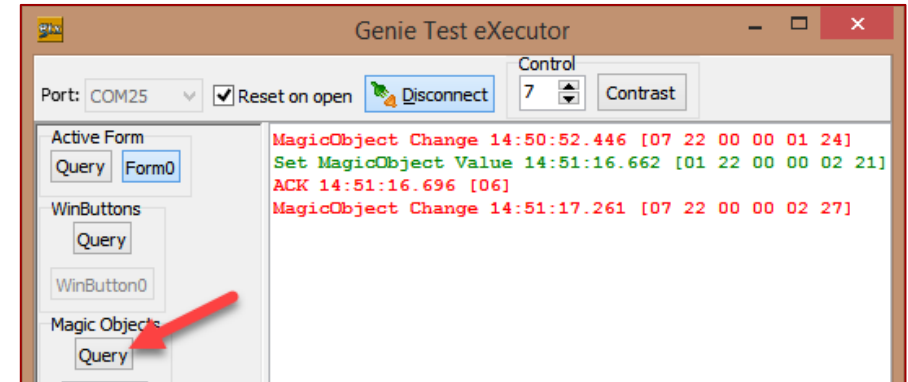
```

MagicObject0.inc
1 //
2 // Added 8/10/2017 9:42:16 AM
3 //
4 func rMagicObject0(var action, var object, var newVal, var *ptr)
5
6     var returnFB := 0;
7
8     if(action == WRITE_OBJ)
9         seroutX(ACK);
10        pause(5);
11        returnFB := flash_Run(newVal);
12
13    if(returnFB)
14        gfx_MoveTo(0,0);
15        if(returnFB == -1)
16            print("invalid bank number");
17        else if(returnFB == -2)
18            print("no valid program in the selected bank");

```

Poll the Current Bank of the Display

Finally, click on the Query button as shown below.



This causes the host to send a READ_OBJ message to the display module. This message polls the bank of the program currently running on the display module.


```

MagicObject Change 14:50:52.446 [07 22 00 00 01 24]
Set MagicObject Value 14:51:16.662 [01 22 00 00 02 21]
ACK 14:51:16.696 [06]
MagicObject Change 14:51:17.261 [07 22 00 00 02 27]
Request MagicObject Value 15:10:54.984 [00 22 00 22]
MagicObject Value 15:10:55.000 [05 22 00 00 02 25]

```

The display replies with a REPORT_OBJ message containing the number of the current bank.

```
MagicObject Change 14:50:52.446 [07 22 00 00 01 24]
Set MagicObject Value 14:51:16.662 [01 22 00 00 02 21]
ACK 14:51:16.696 [06]
MagicObject Change 14:51:17.261 [07 22 00 00 02 27]
Request MagicObject Value 15:10:54.984 [00 22 00 22]
MagicObject Value 15:10:55.000 [05 22 00 00 02 25]
```



The code for sending a REPORT_OBJ message to the host is found on line 28 of MagicObjec0.

```
MagicObject0.inc
25 | endif
26 |
27 | else if(action == READ_OBJ)
28 |     SendReport(REPORT_OBJ, tMagicObject, 0, flash_Bank());
29 | endif
30 |
31 |
32 | func
```

Banks 1 to 5 Demo

The attached zip file “flash1To5Demo.zip” contains demo programs for flash banks 1 to 5. As a whole, the application allows the user to switch between banks 1 to 5, either thru pressing the buttons on the screen or by sending a WRITE_OBJECT message to the serial port. The host can also query the current bank by sending a READ_OBJ message. The programs will also send a REPORT_EVENT message at the start to inform the host of the number of the current bank. Lastly, fade in and out routines were added to the projects. These allow for a visually smoother transition between banks.

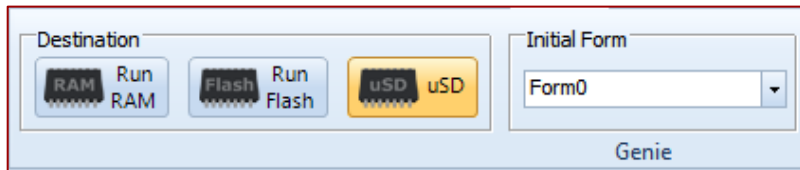
Note: In applications where the programs on the banks contain different objects, the host must be aware which bank is currently active. This way, the host will not communicate with objects that do not exist in the program of the current bank. Otherwise, if the host writes to an object that does not exist in the current program, this will result in data being written to RAM on the display which is either unallocated, or allocated to something else, resulting in failed communications, corruption, or weird behavior.



Important Notes

When using the multiple flash bank feature in ViSi-Genie, there is no ability to pass data between banks. Hence, each bank is a totally separate program. Moreover, switching banks takes a noticeable amount of time, so the host controller must be aware of this and take it into account. Considering the above, it is best to design a multiple flash bank application in such a way that only one bank is used most of the time.

Prior to Worskhop4 version 4.5.0.8, there was no direct way for ViSi-Genie users to utilize flash banks 1 to 5 of a Diablo16 display module. The uSD option in the Destination tab under the Project menu actually required a mother program to be loaded to flash bank 0.



This mother program would then load the child program (the actual ViSi-Genie program) to RAM. Therefore, the actual ViSi-Genie program was not able to utilize any flash space in bank 0 as this was being used by the mother program. Furthermore, the ViSi-Genie program was not able to utilize the whole RAM, since the mother program needed some of it to run and load the ViSi-Genie program itself.

In the new multiple flash bank feature in Worskhop4 version 4.5.0.8 or later, the Update and Bank(s) and Run utility program is loaded to bank 0. ViSi-Genie programs can be separately loaded to the other banks (1 to 5). Therefore, each ViSi-Genie program is able to utilize the entire 32kB flash and 32kB RAM when it runs.

Connect the Display Module to the PC

For instructions on how to save a **ViSi-Genie** project and how to connect the target display to the PC, please refer to the section “**Run the Program**” of the application note

- **First ViSi-Genie Project for Diablo16**

Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.

All trademarks belong to their respective owners and are recognised and acknowledged.

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.