APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi-Genie Magic File Size Request

DOCUMENT DATE: **24th MAY 2019**
DOCUMENT REVISION: **1.1**

Description

This application note primarily shows how the **Magic Object** is used to implement a ViSi-Genie project that allows the host to access files on the uSD card of the display. There are seven types of file access operations:

1. MFILE_READ
2. MFILE_WRITE
3. MFILE_APPEND
4. MFILE_ERASE
5. MFILE_DIR
6. MFILE_SCREEN_CAPTURE
7. MFILE_SIZE

For this application note, the file access operation "MFILE_SIZE" is discussed. The implementation of a file size request operation further requires the use of the following 4DGL features and functions in combination with the **Magic Object**:

- **String class functions**
- **FAT16 file functions**
- **seroutCS(…)**

The **String class functions** and **FAT16 file functions** are functions native to the Picaso and Diablo16 processors.

The function **seroutCS(…)** is one of the Genie Magic callable functions in the ViSi-Genie Communications Protocol. This function writes a parameter to the Genie Serial port and updates the output checksum.

Below is a screenshot image of the project used in this application note.

Note 1: The ViSi-Genie project for this application note is the demo "**FileAccess**", which is found in Workshop. Go to the File menu -> Samples -> ViSi Genie Magic (Picaso/Diablo16) -> **FileAccess.4DGenie**.

Note 2: Workshop Pro is needed for this application.

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and other superseded modules which support the ViSi Genie environment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

Description 2
Content 4
Application Overview 5
Setup Procedure 5
Create a New Project 5
Create a New Project 5
Design the Project 5
Add Two Static Text Objects to Form0 5
Add a Magic Object to Form0 6
Model 6
General Model 6
File Size Request 7
WRITE_MAGIC_BYTES 7
REPORT_MAGIC_EVENT_BYTES 9
File Error 9
The Magic Object 10
Diagram A. Implementation of the General Model Using a Magic Object 11
Diagram B. File Size Request Only Using a Magic Object 11
Is the Message a WRITE_MAGIC_BYTES Message? 11
Parse the Array for the Filename 11
Start Constructing the REPORT_MAGIC_EVENT_BYTES Message 12
Extract the Command Byte 12
Is cmd Equal to “MFILE_SIZE”? 12
Diagram C. File Size Request Operation (Detailed) 12
Open the File 12
Check for Error 13
Get the File Size 13
Close the File Size 13
Send the File Size to the Serial Port 13
Build and Upload the Project 13
Identify the Messages 14
Use the GTX Tool to Analyse the Messages 14
Launch the GTX Tool 14
File Size Request 15
WRITE_MAGIC_BYTES Message 15
REPORT_MAGIC_EVENT_BYTES Message 16
File Error 17
Acknowledgment Byte 17
Proprietary Information 18
Disclaimer of Warranties & Limitation of Liability 18

Application Overview

In the past it was not possible for a host to access files stored on the uSD card of a display module loaded with a ViSi-Genie application. With Workshop 4 Pro it is now possible to accomplish this through the use of the Magic Object. The Magic Object is one of the objects available under the Genie Magic pane. It is actually a 4DGL function which allows users to program the display to handle bytes received from an external host. The user, for instance, can create a Magic Object that waits for a certain sequence of bytes coming from the host. The sequence can contain an instruction byte (e.g. a file size request) and a null-terminated 8.3 format filename (e.g. "datalog1.txt"). Upon receiving this the display module will get the size of the file and send it back to the host. The ViSi-Genie example project "**FileAccess.4DGenie**" is an implementation of the above application.

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).

Create a New Project

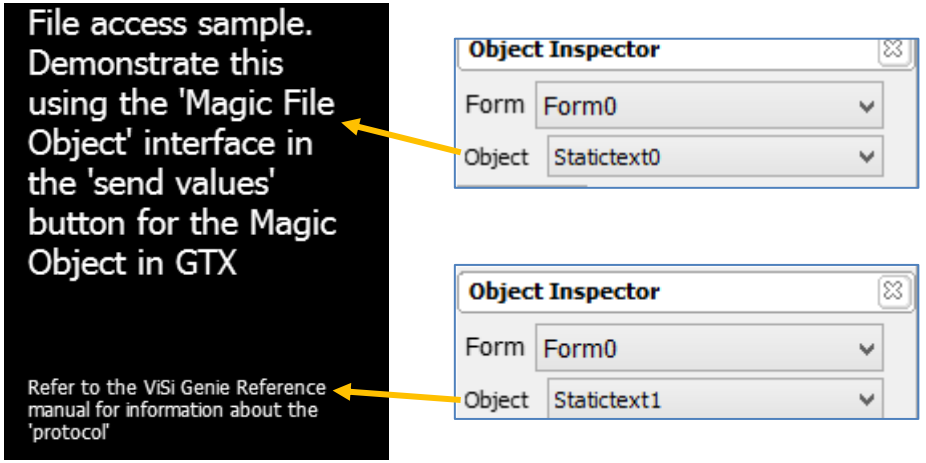
Create a New Project

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

Design the Project

Add Two Static Text Objects to Form0

Two static text objects are added to Form0. These are **Statictext0** and **Statictext1**.

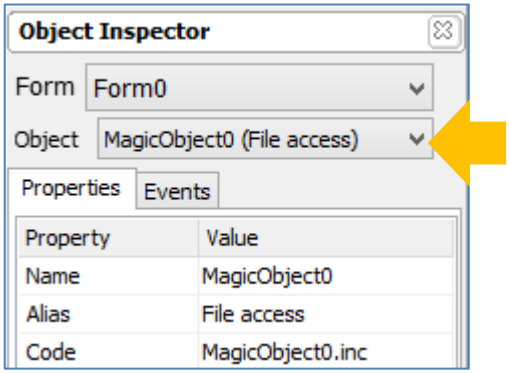


To know more about static text objects, their properties, and how they are added to a project, refer to the application note

ViSi-Genie Labels, Text, and Strings

Add a Magic Object to Form0

A Magic Object is added to **Form0**. This is **MagicObject0**.



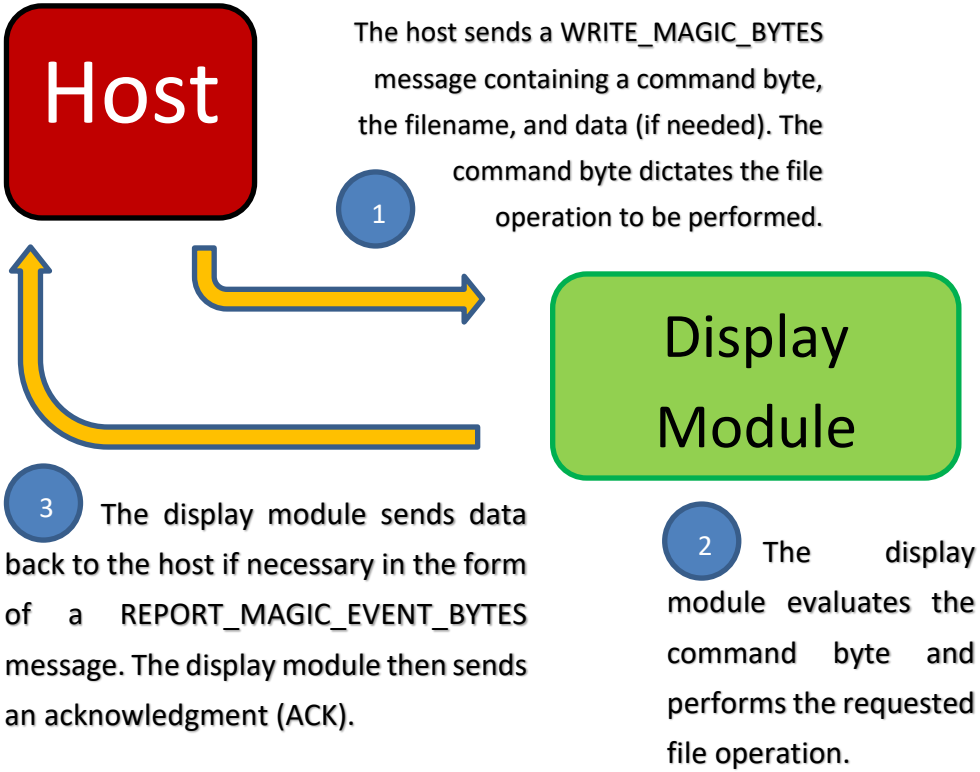
To know more about Magic Objects, their properties, and how they are added to a project, refer to the application note

ViSi-Genie How to Add Magic Objects

Model

General Model

Below is a general model for an application that performs file access operations.

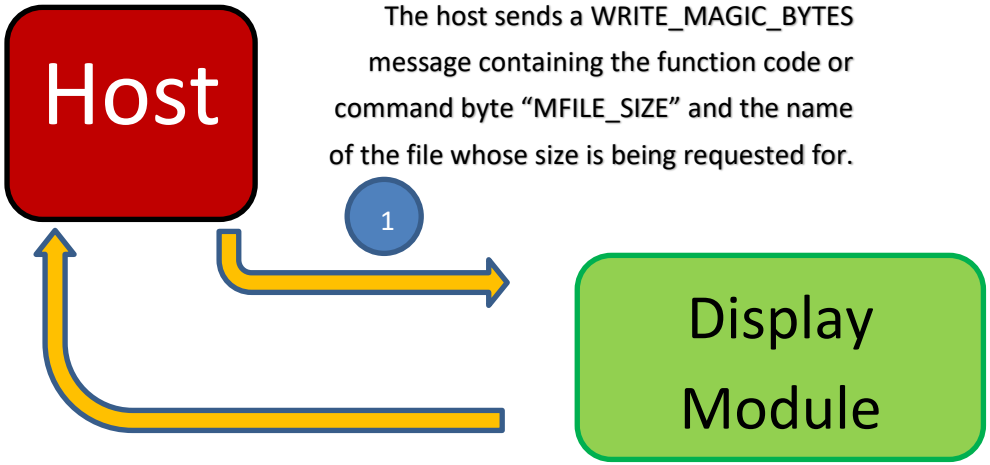


The **WRITE_MAGIC_BYTES** and **REPORT_MAGIC_EVENT_BYTES** messages or commands are two complementary messages that are used in ViSi-Genie Magic. The host sends a **WRITE_MAGIC_BYTES** message, and the display

module, after performing the requested operation, replies with a **REPORT_MAGIC_EVENT_BYTES** message. Steps 2 and 3 can be implemented using a Magic Object.

File Size Request

Below is a model specific to an application that performs a file size request operation.



Section **5.4 (Genie Magic File Access object)** of the ViSi-Genie Reference Manual documents the seven file operations implemented in the example project "**FileAccess.4DGenie**". For this application note, will take look at **MFILE_SIZE**.

WRITE_MAGIC_BYTES

The standard format of WRITE_MAGIC_BYTES message, as defined in section **2.1.2 (Command and Parameters Table)** of the ViSi-Genie Reference Manual is:

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
WRITE_ MAGIC_ BYTES	0x08	Object Index	Length	Array (1 byte values)			Checksum

The section "**2.1.3.8 Write Magic Bytes**" further says:

Description
This command can be used to send an array of bytes to a magic object. The magic object can process the bytes in any way you want it to as there is no restrictions on the format of the information sent. **Note1:** The maximum number of bytes that can be sent at once is set by the 'Maximum String Length' setting in Workshop under File, Options, Genie. **Note2:** A Workshop PRO license is required to use this capability.

Section **5.4 (Genie Magic File Access object)** of the ViSi-Genie Reference Manual describes the WRITE_MAGIC_BYTES message for a file size request operation (as expected by **FileAcces.4DGenie**).

Function	Byte Value	Description and notes	Parameters	Response
MFILE_SIZE	6	Report the size of a file	Function code, Filename	Null, or 4 bytes

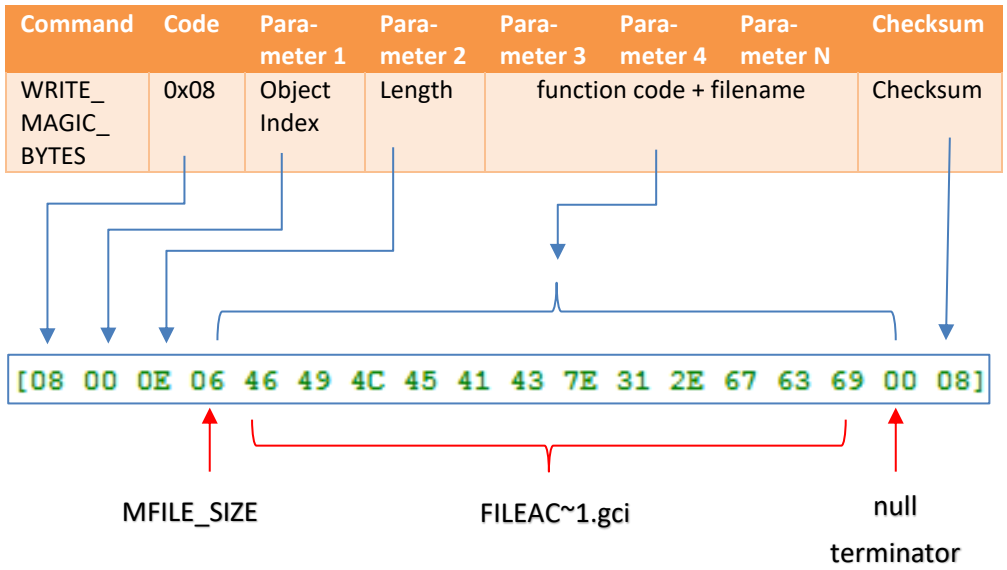
Thus, a WRITE_MAGIC_BYTES message specific to a file size request operation, has the format:

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
WRITE_ MAGIC_ BYTES	0x08	Object Index	Length	function code + filename			Checksum

To request for the size of the file "**FILEAC~1.gci**", the host would send:

```
[08 00 0E 06 46 49 4C 45 41 43 7E 31 2E 67 63 69 00 08]
```

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
WRITE_ MAGIC_ BYTES	0x08	Object Index	Length	function code + filename			Checksum



REPORT_MAGIC_EVENT_BYTES

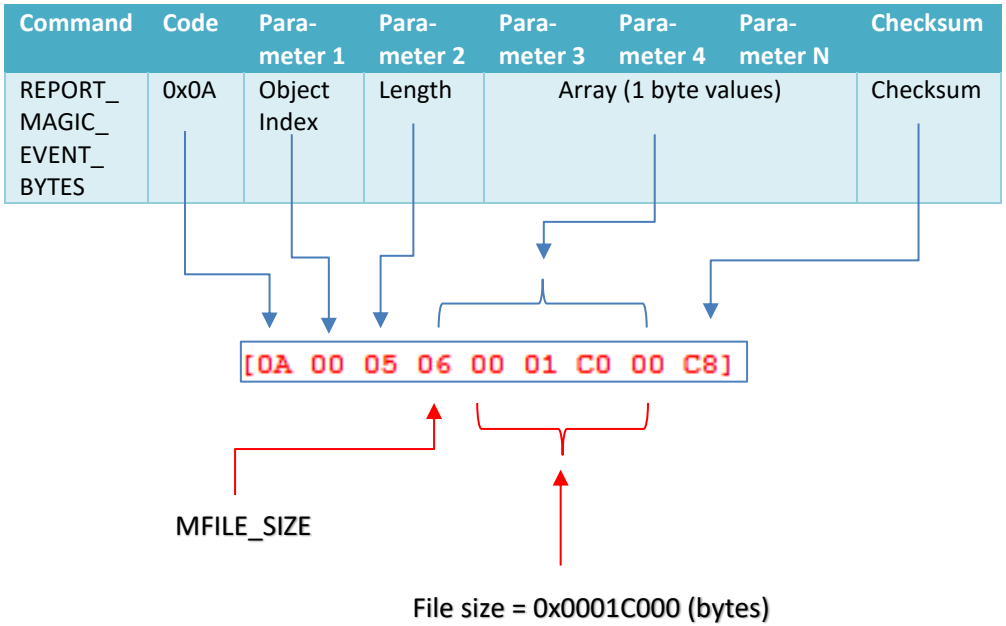
The standard format of a REPORT_MAGIC_EVENT_BYTES message, as defined in section **2.1.2 (Command and Parameters Table)** of the ViSi-Genie Reference Manual is:

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
REPORT_ MAGIC_ EVENT_ BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum

Section **5.4 (Genie Magic File Access object)** of the ViSi-Genie Reference Manual states that a REPORT_MAGIC_EVENT_BYTES message for a file size request operation, as implemented in **FileAcces.4DGenie**, will have the following additional data.

Command(s)	Type	Notes
Size	Function code	MFILE_SIZE
	'Null'	Only the 'function code' will be sent if an error occurs (eg file does not exist)
	4 Bytes	File size big Endian (i.e. most significant byte first)

Below is an example of a REPORT_MAGIC_EVENT_BYTES message for a successful file size request operation.



File Error

If an error occurs during the operation, an empty or null REPORT_MAGIC_EVENT_BYTES message is sent back by the display module. The message will contain only the command byte for file size request.

The Magic Object

Going back to our working model for a file size request operation, the host would need to send a WRITE_MAGIC_BYTES message to the display module (step 1). The display module then acquires the file size (step 2) and sends back a REPORT_MAGIC_EVENT_BYTES message, along with an ACK byte (step 3).

We have also seen that the demo "**FileAccess.4DGenie**" expects the host to follow a certain format for a WRITE_MAGIC_BYTES message. The demo "**FileAccess.4DGenie**" also follows a certain format when it constructs a REPORT_MAGIC_EVENT_BYTES message to be sent back to the host. These formats are in addition to the standard formats of WRITE_MAGIC_BYTES and REPORT_MAGIC_EVENT_BYTES messages described in the ViSi-Genie Reference Manual.

The demo "**FileAccess.4DGenie**" uses a **Magic Object** to receive and handle WRITE_MAGIC_BYTES messages, to perform the requested operation, and to send REPORT_MAGIC_EVENT_BYTES messages.

The prototype of the 4DGL function inside a **Magic Object** is:

rMagicObject0(var action, **var** object, **var** newVal, **var** *ptr**);**

Since this function will be used to receive and handle the WRITE_MAGIC_BYTES message coming from the host, it is expected that the passed parameters will give the user access to the received WRITE_MAGIC_BYTES message. Below are the descriptions of the parameters, as per the section "**5.1 Genie Magic callable Functions**" of the ViSi-Genie Reference Manual.

Parameter	Description
action	The command that was received from the host, one of 1. READ_OBJ, 2. WRITE_OBJ, 3. WRITE_MAGIC_BYTES or 4. WRITE_MAGIC_DBYTES
object	Normally the object received from the host will be the same as the **n** in the function name, but since you could call this function internally it might be something else.
newVal	N/A for Action of READ_OBJ, New value for Action of WRITE_OBJ, Otherwise number of parameters in the ptr array.
ptr	N/A for Action of READ_OBJ and WRITE_OBJ, otherwise Pointer to array of parameters passed. Array is always a standard Picaso/Diablo integer array. For WRITE_MAGIC_BYTES each element contains a byte.

For the example project "**FileAccess.4DGenie**", the parameter "**action**" is **WRITE_MAGIC_BYTES**. The parameter "**object**" is **MagicObject0**. The parameter "**newVal**" is the length of the array or the combined length of the command byte and the filename string. The parameter "**ptr**" is a pointer to the array which will contain the data from the host.

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
WRITE_ MAGIC_ BYTES	0x08	Object Index	Length	Array (1 byte values)			Checksum

rMagicObject0(var action, **var** object, **var** newVal, **var** *ptr**);**

Diagram A. Implementation of the General Model Using a Magic Object

Attached is the PDF file "**programFlow.pdf**". It contains three diagrams, the first of which illustrates the implementation of the general model. This diagram represents the example project "**FileAccess.4DGenie**". The area bounded by the broken lines is implemented using a Magic Object.

Diagram B. File Size Request Only Using a Magic Object

The second diagram of the PDF file "**programFlow.pdf**" represents the scope of this application note – the file size request operation.

Is the Message a WRITE_MAGIC_BYTES Message?

```
if (action == WRITE_MAGIC_BYTES)
```

Parse the Array for the Filename

The array pointed to by **ptr** is an array composed of 16-bit elements. The filename is to be extracted from this array. In 4DGL, characters can be stored as 16-bit elements in an array (word-aligned) or as a string (byte-aligned). The string class functions apply only to strings. To illustrate using the filename "ascii.txt":

16-bit element array

address	ptr[1]	ptr[2]	ptr[3]	ptr[4]	ptr[5]
Content	0x0061	0x0073	0x0063	0x0069	0x0069
char	a	s	c	i	i

ptr[6]	ptr[7]	ptr[8]	ptr[9]	ptr[10]
0x002E	0x0074	0x0078	0x0074	0x0000
.	t	x	t	null

4DGL string

address	ptr[1]	ptr[2]	ptr[3]	ptr[4]	ptr[5]
Content	0x7361	0x6963	0x2E69	0x7874	0x0074
char	sa	ic	.i	xt	nullt

Note the difference in endianness and manner of storage. The message received from the host is stored in the array pointed to by **ptr.** This array is internal to Genie and is word-aligned. Since the demo "**FileAccess.4DGenie**" uses string class functions to operate on the filename, there is therefore a

need to convert the 16-bit element array containing the filename to a 4DGL string. Hence the routine

```
for (i := 1; i < newVal; i++)                    // cha
    j := i*2 ;
    ptr[i] := (ptr[j] << 8) + ptr[j-1] ;
next
```

The 4DGL string class operations can now be used to operate on or manipulate the filename converted as a 4DGL string. Prior to this however, a string pointer to the filename must be defined.

```
fname := str_Ptr(&ptr[1]) ;                      // bui
```

For more information on 4DGL strings, please refer to the Picaso or Diablo16 Internal Functions Reference Manual. Right-click on a string class function name text and choose "Context Sensitive help" to open the manual. Also, the application note below discusses strings in 4DGL.

Designer or ViSi String Class Function

Start Constructing the REPORT_MAGIC_EVENT_BYTES Message

```
seroutCS(REPORT_MAGIC_EVENT_BYTES) ;     // we
seroutCS(object) ;
```

To know more about the function **"seroutCS(…)"**,see **section 5.1 Genie Magic callable Functions** of the ViSi-Genie Reference Manual.

Extract the Command Byte

```
cmd    := ptr[0] ;                                // ext
```

Is cmd Equal to "MFILE_SIZE"?

```
switch (cmd)

  case MFILE_SIZE :
```

Diagram C. File Size Request Operation (Detailed)

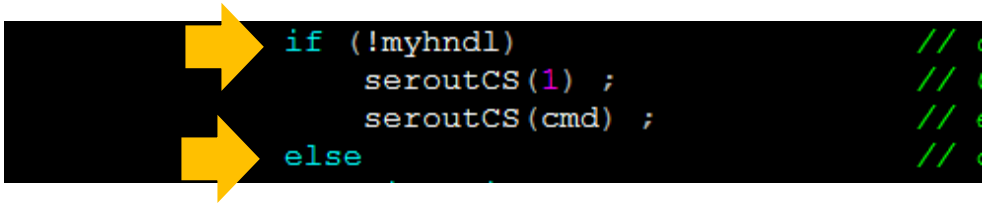
The third diagram of the PDF file "**programFlow.pdf**" presents a more detailed view of the file size request operation.

Open the File

```
myhndl := file_Open(fname, 'a') ;
```

The function "**file_Open(…)**" is one of the several FAT16 file functions in 4DGL. FAT16 file functions are used mainly for accessing and modifying files on a FAT16-formatted uSD card. For more information on the FAT16 file functions, please refer to the Picaso or Diablo16 Internal Functions Reference Manual. Right-click on a FAT16 file function name text and choose "Context Sensitive help" to open the manual.

Check for Error



```
if (!myhndl)
    seroutCS(1) ;
    seroutCS(cmd) ;
else
```

Get the File Size

```
file_Size(myhndl,&szh, &szl) ;
```

Close the File Size

```
file_Close(myhndl) ;
```

The functions **"file_Size (…)"** and **"file_Close(…)"** are examples of string class functions in 4DGL. These functions are used mainly for evaluating and manipulating strings. For more information on the FAT16 file functions, please refer to the Picaso or Diablo16 Internal Functions Reference Manual. Right-click on a FAT16 file function name text and choose "Context Sensitive help" to open the manual.

Send the File Size to the Serial Port

```
seroutCS(szh >> 8);
seroutCS(szh);
seroutCS(szl >> 8);
seroutCS(szl);
```

Note that the size of a file is a 32-bit integer value.

Build and Upload the Project

For instructions on how to build and upload a ViSi-Genie project to the target display, please refer to the section "**Build and Upload the Project**" 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).

The uLCD-32PTU and/or the uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

Identify the Messages

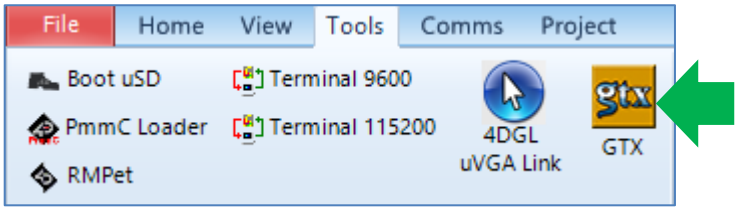
The display module is going to send and receive messages to and from an external host. This section explains to the user how to interpret these messages. An understanding of this section is necessary for users who intend to interface the display to a host. The ViSi Genie Reference Manual is recommended for advanced users.

Use the GTX Tool to Analyse the Messages

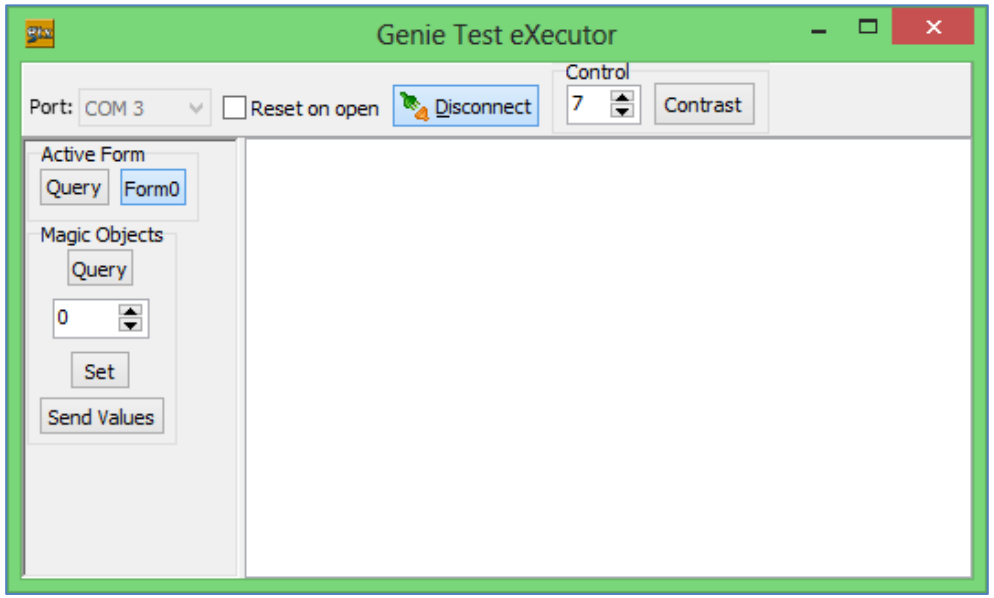
Using the GTX or **Genie Test eXecutor** tool is one option to get the messages sent by the display to the host. Here the PC will be the host. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to the display module. It is an essential debugging tool.

Launch the GTX Tool

Under the Tools menu click on the GTX tool button.



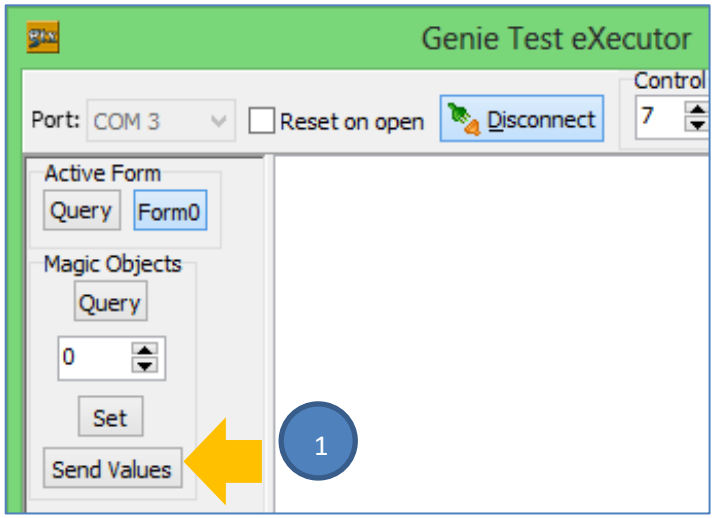
The Genie Test eXecutor window appears.

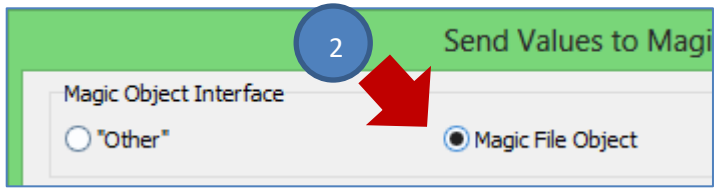


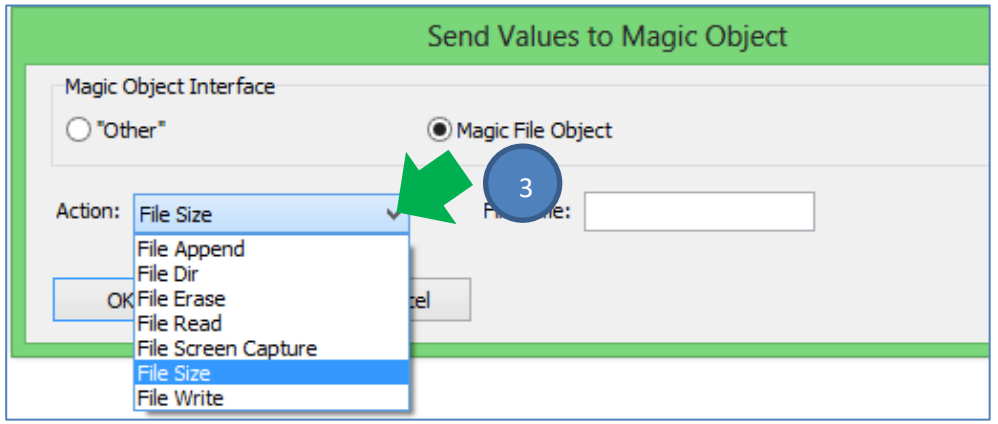
File Size Request

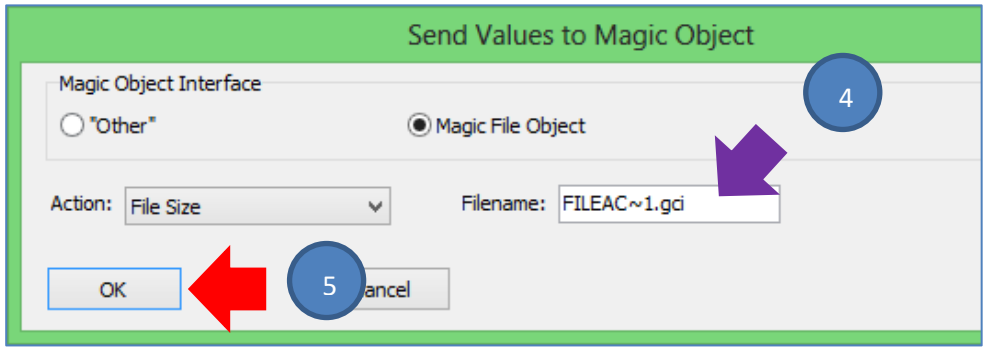
WRITE_MAGIC_BYTES Message

Send the MFILE_SIZE command and the filename in a WRITE_MAGIC_BYTES message.









Note: **"FILEAC~1.gci"** is the filename of the generated graphics file for the Workshop example project **"FileAccess.4DGenie"**. When you build this project, Workshop generates the supporting files and their 8.3 format-filenames and copies them to the uSD card. FILEAC~1.gci is one of these files. If the name of the project is different or was changed, replace "FILEAC~1.gci" with the correct or another filename. Make sure that the file exists on the uSD card.

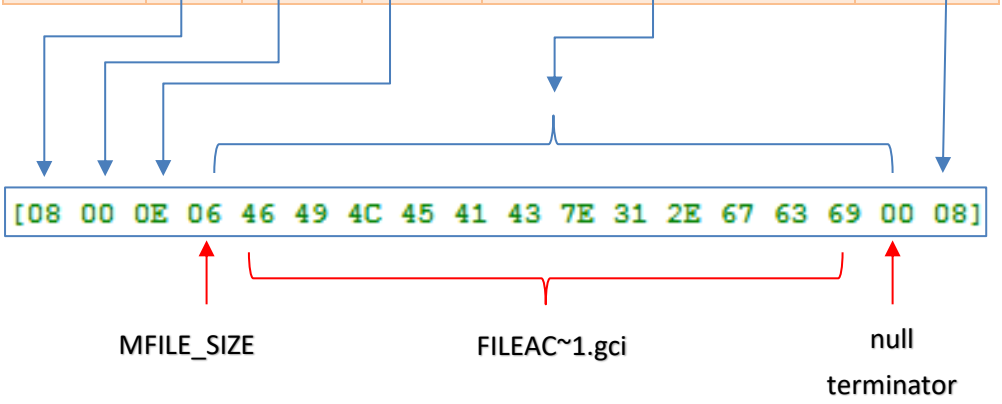
The GTX tool sends the WRITE_MAGIC_BYTES message.

```
Set MagicObject byte Value 15:00:43.459
```

```
[08 00 0E 06 46 49 4C 45 41 43 7E 31 2E 67 63 69 00 08]
```

The format of this message is:

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
WRITE_ MAGIC_ BYTES	0x08	Object Index	Length	function code + filename			Checksum



REPORT_MAGIC_EVENT_BYTES Message

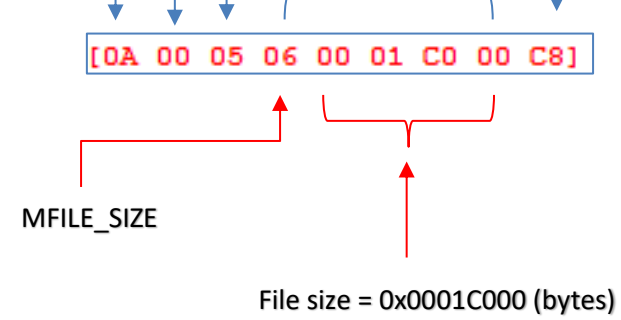
The display module replies with a REPORT_MAGIC_EVENT_BYTES message containing the 32-bit file size.

```
Magic Object Change Bytes 15:00:43.538
```

```
[0A 00 05 06 00 01 C0 00 C8]
```

The format of this message is:

Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
REPORT_ MAGIC_ EVENT_ BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum



File Error

If the file were not present on the uSD card, the display module would return a REPORT_MAGIC_EVENT_BYTES message that contains only the function code "MFILE_SIZE".

```
Magic Object Change Bytes 16:20:26.231 [0A 00 01 06 0D]
```

Acknowledgment Byte

Either way the display module replies with an acknowledgement byte.

```
ACK 11:13:21.877 [06]
```

Proprietary Information

Disclaimer of Warranties & Limitation of Liability