



# ViSi Genie Magic How to read a File

DOCUMENT DATE: **8<sup>th</sup> MAY 2020**  
DOCUMENT REVISION: **1.2**



## 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\_READ” is discussed. The implementation of a file read operation further requires the use of the following 4DGL features and functions in combination with the **Magic Event** 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 display modules:

[uLCD-24PTU](#)      [uLCD-28PTU](#)      [uVGA-III](#)  
[gen4-uLCD-24PT](#)    [gen4-uLCD-28PT](#)    [gen4-uLCD-32PT](#)

and other superseded modules which support the ViSi Genie environment.

- The target module can also be a Diablo16 display

[gen4-uLCD-24D](#)      [gen4-uLCD-28D](#)      [gen4-uLCD-32D](#)  
  Series                    Series                    Series  
[gen4-uLCD-35D](#)      [gen4-uLCD-43D](#)      [gen4-uLCD-50D](#)  
  Series                    Series                    Series  
[gen4-uLCD-70D](#)  
  Series  
[uLCD-35DT](#)      [uLCD-43D Series](#)      [uLCD-70DT](#)

Visit [www.4dsystems.com.au/products](http://www.4dsystems.com.au/products) to see the latest display module products that use the Diablo16 processor.

- [4D Programming Cable /  \$\mu\$ USB-PA5/ \$\mu\$ USB-PA5-II](#) for non-gen4 displays (uLCD-xxx)
- [4D Programming Cable](#) & [gen4-IB](#) / [gen4-PA](#) / [4D-UPA](#), for gen-4 displays (gen4-uLCD-xxx)
- [micro-SD \( \$\mu\$ SD\)](#) 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

<b>Description</b> .....	<b>2</b>
<b>Content</b> .....	<b>4</b>
<b>Application Overview</b> .....	<b>5</b>
<b>Setup Procedure</b> .....	<b>5</b>
<b>Create a New Project</b> .....	<b>6</b>
<i>Create a New Project</i> .....	<b>6</b>
<b>Design the Project</b> .....	<b>6</b>
<i>Add Two Static Text Objects to Form0</i> .....	<b>6</b>
<i>Add a Magic Object to Form0</i> .....	<b>6</b>
<i>Model</i> .....	<b>7</b>
File Access Operations .....	<b>7</b>
File Read .....	<b>7</b>
WRITE_MAGIC_BYTES .....	<b>8</b>
REPORT_MAGIC_EVENT_BYTES .....	<b>9</b>
File Read Error .....	<b>10</b>
Small Files .....	<b>10</b>
Large Files .....	<b>11</b>
Large Files – 1 <sup>st</sup> Packet .....	<b>12</b>
Large Files – 2 <sup>nd</sup> Packet .....	<b>12</b>
<i>The Magic Object</i> .....	<b>13</b>

<i>Diagram A. Implementation of the General Model Using a Magic Object</i> .....	<b>14</b>
<i>Diagram B. Implementation of the File Access Model Using a Magic Object</i> .....	<b>14</b>
Is the Message a WRITE_MAGIC_BYTES Message? .....	<b>14</b>
Parse the Array for the Filename .....	<b>14</b>
Start Constructing the REPORT_MAGIC_EVENT_BYTES Message .....	<b>15</b>
Extract the Command Byte .....	<b>15</b>
Is cmd Equal to “MFILE_READ”? .....	<b>15</b>
<i>Diagram C. File Read Operation</i> .....	<b>15</b>
Open the File .....	<b>15</b>
Check for Error .....	<b>15</b>
Get the File Size .....	<b>16</b>
Get a Byte from the File then Send it to the Serial Port .....	<b>16</b>
<b>Build and Upload the Project</b> .....	<b>16</b>
<b>Copy the uSD Card Files</b> .....	<b>16</b>
<b>Identify the Messages</b> .....	<b>17</b>
<i>Use the GTX Tool to Analyse the Messages</i> .....	<b>17</b>
Launch the GTX Tool .....	<b>17</b>
<i>The File does not Exist</i> .....	<b>17</b>
WRITE_MAGIC_BYTES Message .....	<b>17</b>
REPORT_MAGIC_EVENT_BYTES Message .....	<b>19</b>
<i>The File is Less than 253 Bytes in Size</i> .....	<b>19</b>

WRITE_MAGIC_BYTES Message	19
REPORT_MAGIC_EVENT_BYTES Message	21
<i>The File is more than 253 Bytes in Size</i> .....	21
WRITE_MAGIC_BYTES Message	21
REPORT_MAGIC_EVENT_BYTES Message	22
<b>Proprietary Information</b> .....	24
<b>Disclaimer of Warranties &amp; Limitation of Liability</b> .....	24

## Application Overview

In the past it was not possible for a host to access files stored in 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 14 bytes from the host. The 14 bytes can contain an instruction byte, followed by a null-terminated 8.3 format filename (e.g. "datalog1.txt"). The instruction byte can be a file read, a file write, a file append, etc. Upon receiving a file read instruction together with the filename, for example, the display module will send back the contents of the file (if it exists) to the host. The ViSi-Genie example project "**FileAccess.4DGenie**" is an example 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)

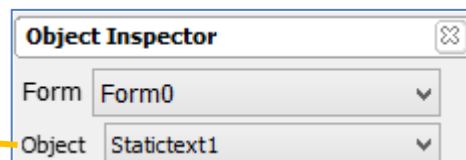
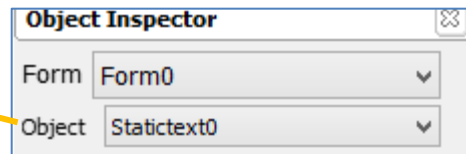
## Design the Project

### Add Two Static Text Objects to Form0

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

File access sample.  
Demonstrate this using the 'Magic File Object' interface in the 'send values' button for the Magic Object in GTX

Refer to the ViSi Genie Reference manual for information about the 'protocol'

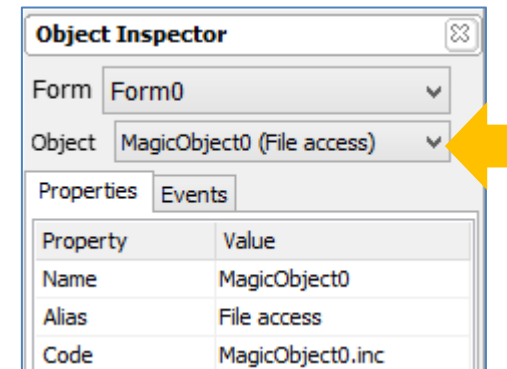


To know more about static text objects, their properties, and how they are added to a project, refer to the application note

[ViSi-Genie Labels, Text, and Strings](#)

### Add a Magic Object to Form0

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



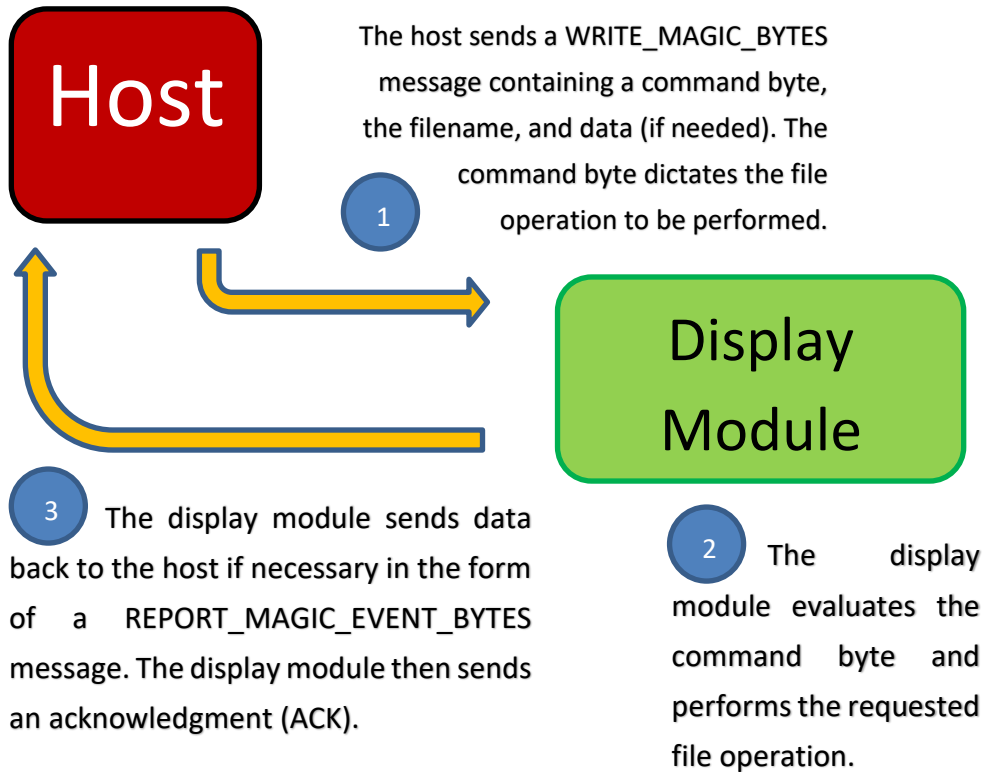
To know more about Magic Objects, their properties, and how they are added to a project, refer to the application note

[ViSi-Genie How to Add Magic Objects](#)

## Model

### File Access Operations

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

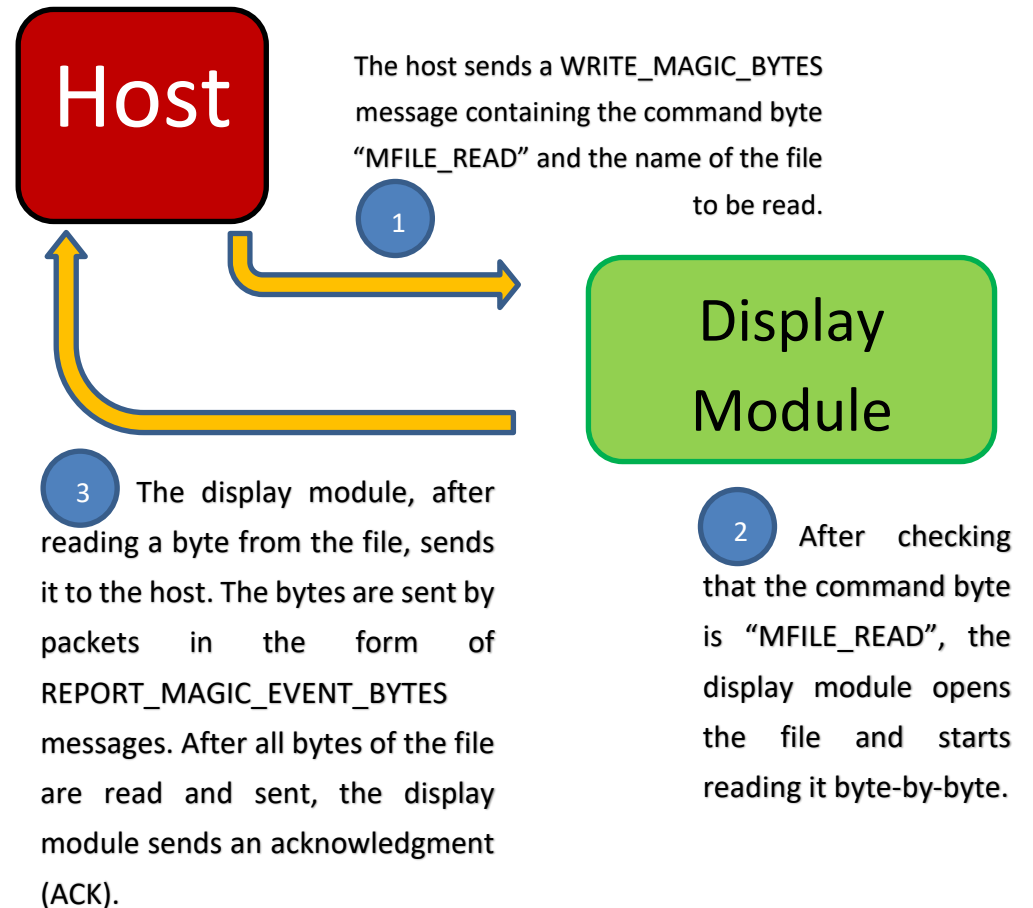


The `WRITE_MAGIC_BYTES` and `REPORT_MAGIC_EVENT_BYTES` messages or commands are two complementary messages that are used in ViSi-Genie Magic. The host sends a `WRITE_MAGIC_BYTES` message, and the display module, after performing the requested operation, replies with a

`REPORT_MAGIC_EVENT_BYTES` message. Steps 2 and 3 can be implemented using a Magic Object.

### File Read

Below is a model specific to an application that performs a file read 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\_READ**.

**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

Section 5.4 (Genie Magic File Access object) of the ViSi-Genie Reference Manual describes the WRITE\_MAGIC\_BYTES message for a file read operation (as expected by FileAcces.4DGenie).

Function	Byte Value	Description and notes	Parameters	Response
<b>MFILE_READ</b>	0	Read a file. This reads the entire file.	Function code, Filename	More/final, bytes

Thus, a WRITE\_MAGIC\_BYTES message specific to a file read operation, has the format:

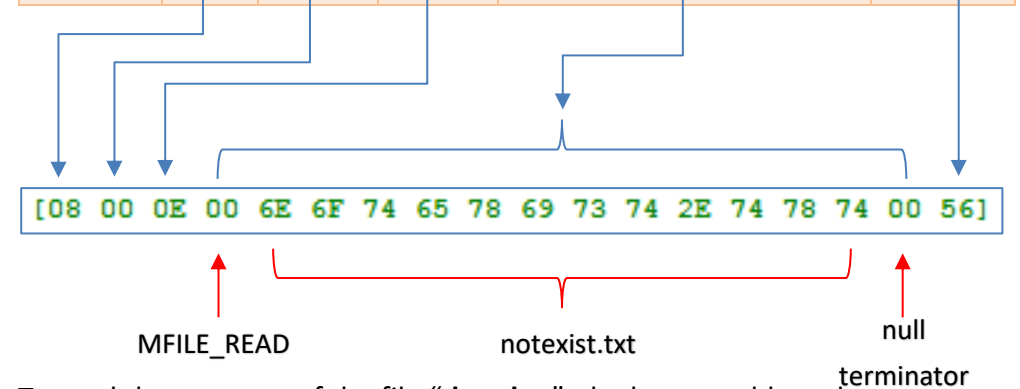
Command	Code	Para- meter 1	Para- meter 2	Para- meter 3	Para- meter 4	Para- meter N	Checksum
WRITE_MAGIC_BYTES	0x08	Object Index	Length	Function code + filename			Checksum

Let us look at two examples of a WRITE\_MAGIC\_BYTES message sent from a host to the display for a file read operation.

To read the contents of the file “notexist.txt”, the host would send:

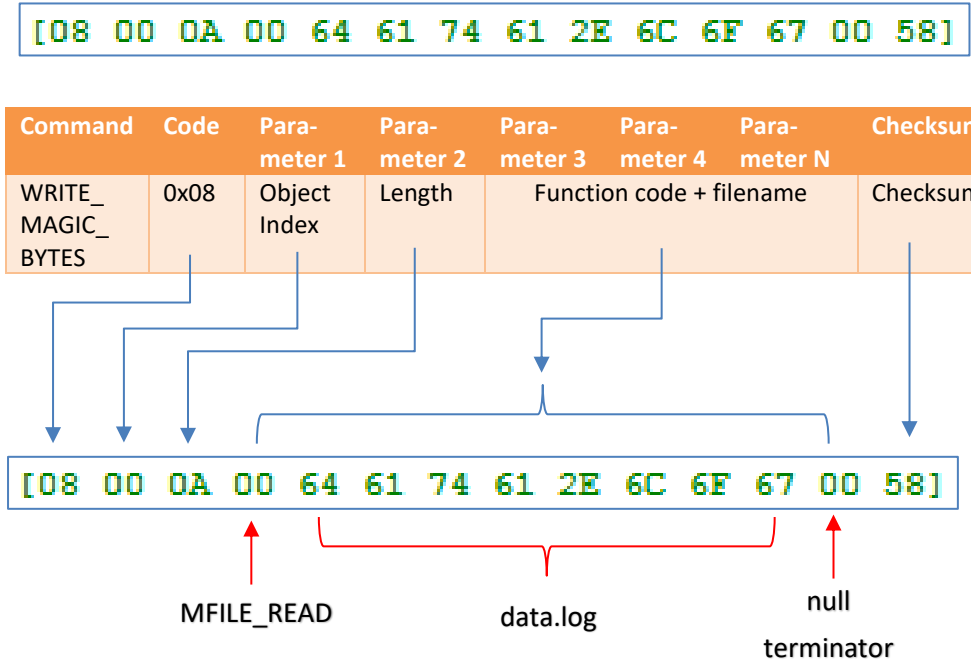
```
[08 00 0E 00 6E 6F 74 65 78 69 73 74 2E 74 78 74 00 56]
```

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 read the contents of the file “data.log”, the host would send:





**REPORT\_MAGIC\_EVENT\_BYTES**

The standard format of REPORT\_MAGIC\_EVENT\_BYTES message, as defined in section 2.1.2 (Command and Parameters Table) of the ViSi-Genie Reference Manual is:

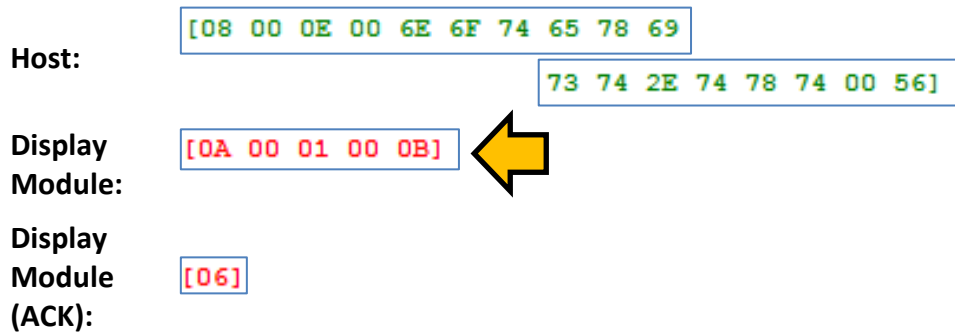
Command	Code	Para- meter 1	Para- meter 2	Para- meter 3	Para- meter 4	Para- meter N	Checksum
REPORT_ MAGIC_ EVENT_ BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum

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 read operation, as implemented in FileAcces.4DGenie, will have the following additional data.

Command(s)	Type	Notes
Read	Function code	MFILE_READ
	'Null'	Only the 'function code' will be sent if an error occurs (eg file does not exist)
	More / Final byte	0x00 indicates this is the last data for the file, 0xFF indicates another message follows this one.
	Bytes	Up to 255 bytes will be in each message. i.e. up to 253 bytes of file data

**File Read Error**

If an error occurs during the file read operation (e.g. the file to be read does not exist), 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 read. For example, if the file “notexist.txt” does not exist, the display module would reply with the message shown below, followed by an acknowledgment byte (ACK).



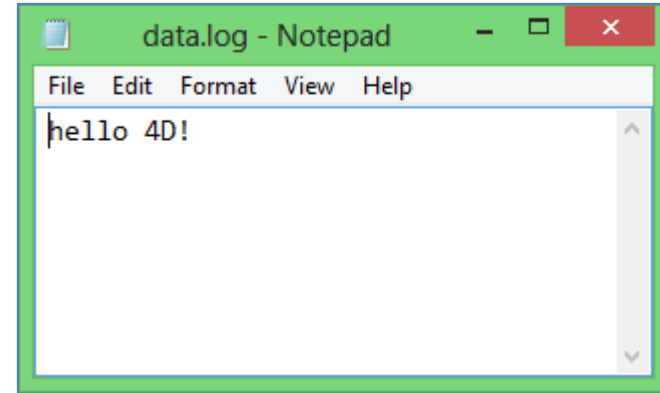
Command	Code	Para- meter 1	Para- meter 2	Para- meter 3	Para- meter 4	Para- meter N	Checksum
REPORT_MAGIC_EVENT_BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum

[0A 00 01 00 0B]

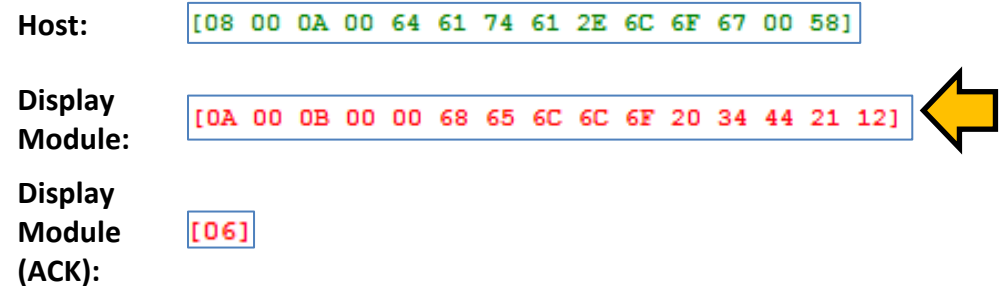
↑  
MFILE\_READ

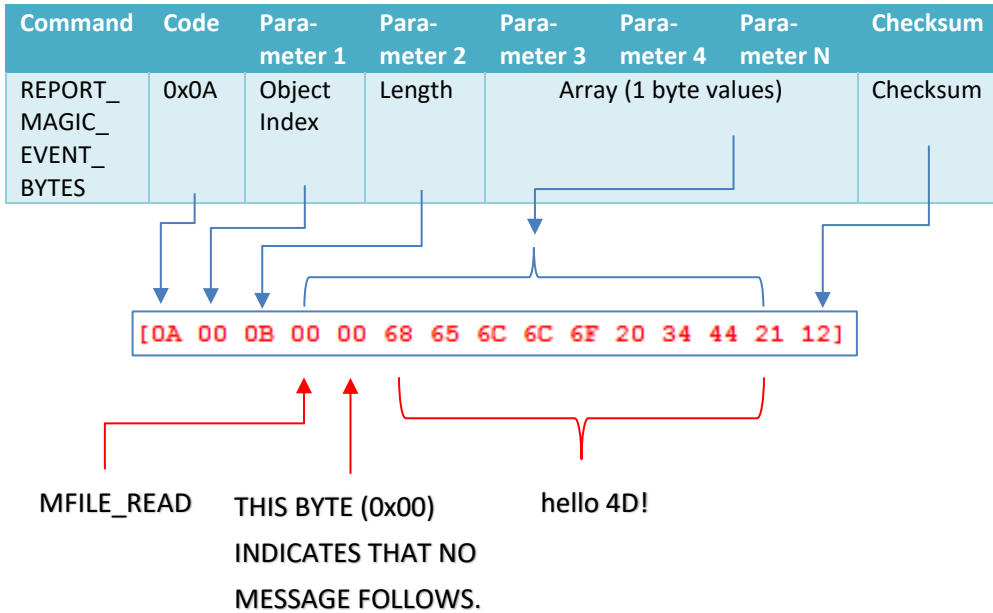
**Small Files**

For this case we assume that the text file “data.log” exists on the uSD card of the display module. This text file contains a short string less than 253 bytes long.



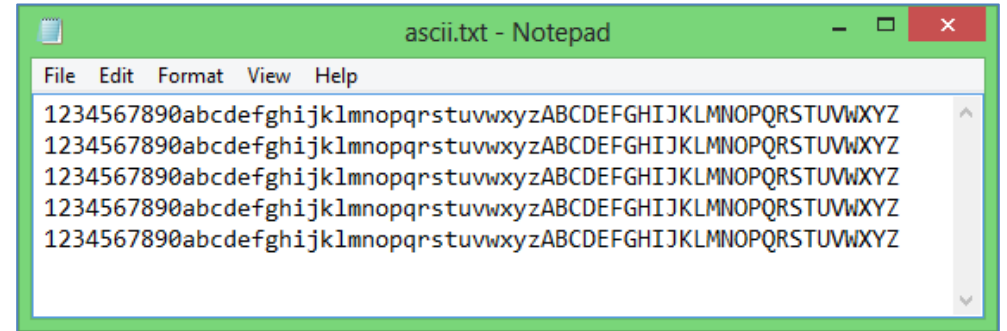
The display module would reply with the message shown below, followed by an acknowledgment byte (ACK).





**Large Files**

For a more reliable data transfer, the contents of a large file being read are divided into packets, each of which is 253 bytes long. Each packet is then inserted into a **REPORT\_MAGIC\_EVENT\_BYTES** message. To illustrate the case for data transfer of large files, we assume that the file “**ascii.txt**” exists on the uSD card. This file contains alphanumeric characters and is 318 bytes long.



The contents of the file will be divided into two packets. The first packet is 253 bytes long, and the second is 65 bytes long. To illustrate,

<b>Host:</b>	[08 00 0B 00 61 73 63 69 69 2E 74 78 74 00 24]
<b>Display Module (1<sup>st</sup> packet):</b>	[0A 00 FF 00 FF 31 32 33 34 35 36 37 38 39 30 6C]
<b>Display Module (2<sup>nd</sup> packet):</b>	[0A 00 43 00 00 5A 0D 0A 31 32 33 34 35 36 37 38]
<b>Display Module (ACK):</b>	[06]

**Large Files - 1<sup>st</sup> Packet**

Note that the screenshot image of the message was cropped.

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

[0A 00 FF 00 FF 31 32 33 34 35 36 37 38 39 30 63]

MFILE\_READ

THIS BYTE (0x00)  
INDICATES THAT NO  
MESSAGE FOLLOWS.

Z..1234567...

**Large Files - 2<sup>nd</sup> Packet**

Note that the screenshot image of the message was cropped.

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

[0A 00 43 00 00 5A 0D 0A 31 32 33 34 35 36 37 33]

MFILE\_READ

THIS BYTE (0x00)  
INDICATES THAT NO  
MESSAGE FOLLOWS.

Z..1234567...

For a successful file read operation, the fifth byte of a REPORT\_MAGIC\_EVENT\_BYTES message indicates if another message follows or not. Using this byte as a flag, the host would know when the data transfer of a large file is complete.

## The Magic Object

Going back to our working model for a file read operation, the host would need to send a `WRITE_MAGIC_BYTES` message to the display module (step 1). The display module then performs a file read operation (step 2) and sends back the contents of the file to the host in the form of `REPORT_MAGIC_EVENT_BYTES` messages, 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:

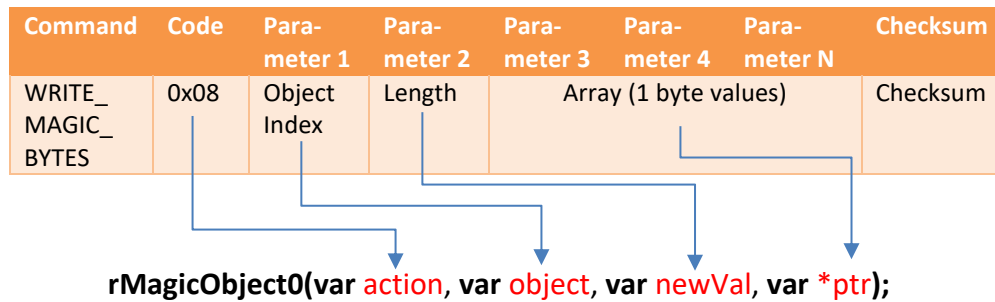
```
rMagicObject0(var action, var object, var newVal, var *ptr);
```

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
<b>action</b>	The command that was received from the host, one of <ol style="list-style-type: none"> <li>1. <code>READ_OBJ</code>,</li> <li>2. <code>WRITE_OBJ</code>,</li> <li>3. <code>WRITE_MAGIC_BYTES</code> or</li> <li>4. <code>WRITE_MAGIC_DBYTES</code></li> </ol>
<b>object</b>	Normally the object received from the host will be the same as the <b>n</b> in the function name, but since you could call this function internally it might be something else.
<b>newVal</b>	N/A for Action of <code>READ_OBJ</code> , New value for Action of <code>WRITE_OBJ</code> , Otherwise number of parameters in the ptr array.
<b>ptr</b>	N/A for Action of <code>READ_OBJ</code> and <code>WRITE_OBJ</code> , otherwise Pointer to array of parameters passed. Array is always a standard Picaso/Diablo integer array. For <code>WRITE_MAGIC_BYTES</code> 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.



**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. Implementation of the File Access Model Using a Magic Object**

The second diagram of the PDF file “**programFlow.pdf**” represents the scope of this application note – the file read 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:

**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 ordinary 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\_READ”?

```
switch (cmd)
  case MFILE_READ :
```

### Diagram C. File Read Operation

The third diagram of the PDF file “[programFlow.pdf](#)” presents a more detailed view of the file read operation.

### Open the File

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

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) // e
  seroutCS(1) ; // e
  seroutCS(cmd) ; // e
else // e
  file_Size(myhndl, szh, szl) ;
```

### Get the File Size

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

The function “**file\_Size(...)**” 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.

### Get a Byte from the File then Send it to the Serial Port

```
seroutCS(file_GetC(myhndl) );
```

The function “**file\_GetC(...)**” 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.

## 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.

## Copy the uSD Card Files

Attached to this application note is the folder “**uSD card files**” which contains two files as shown below. Copy these files to the uSD card before inserting it to the uSD card slot of the display module.

Name	Type
 ascii.txt	Text Document
 data.log	Text Document



## Identify the Messages

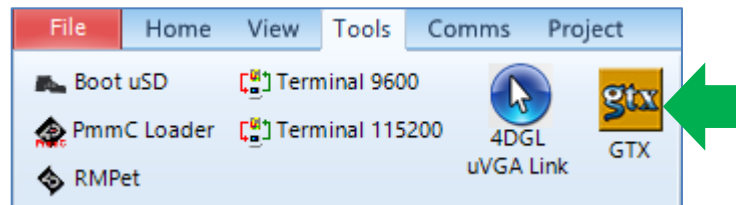
The display module is going to send and receive messages to and from an external host. This section explains to the user how to interpret these messages. An understanding of this section is necessary for users who intend to interface the display to a host. The [ViSi Genie Reference Manual](#) is recommended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

### Use the GTX Tool to Analyse the Messages

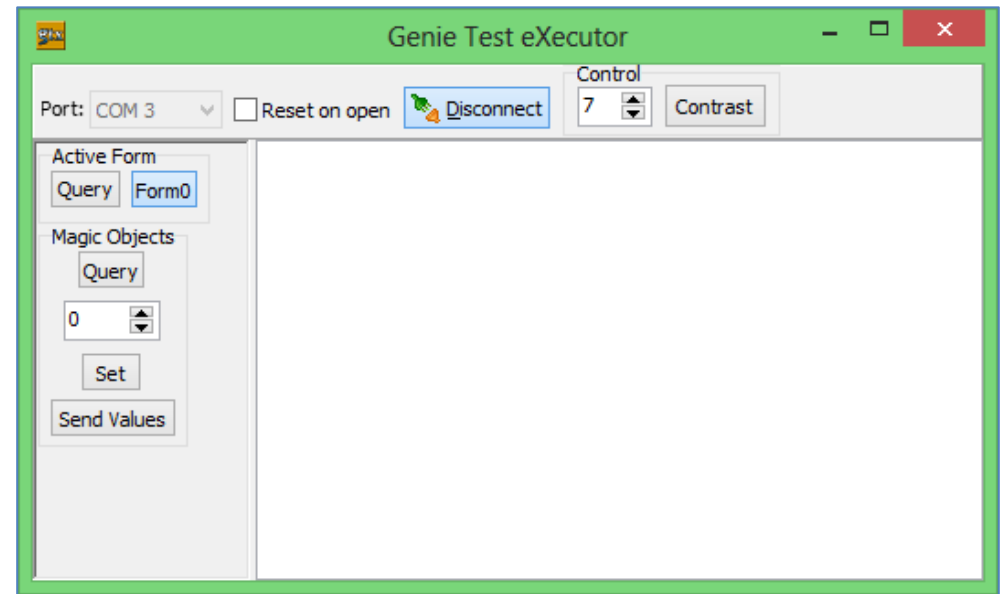
Using the GTX or **Genie Test eXecutor** tool is one option to get the messages sent by the display to the host. Here the PC will be the host. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to the display module. It is an essential debugging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

### Launch the GTX Tool

Under the Tools menu click on the GTX tool button.



The Genie Test eXecutor window appears.



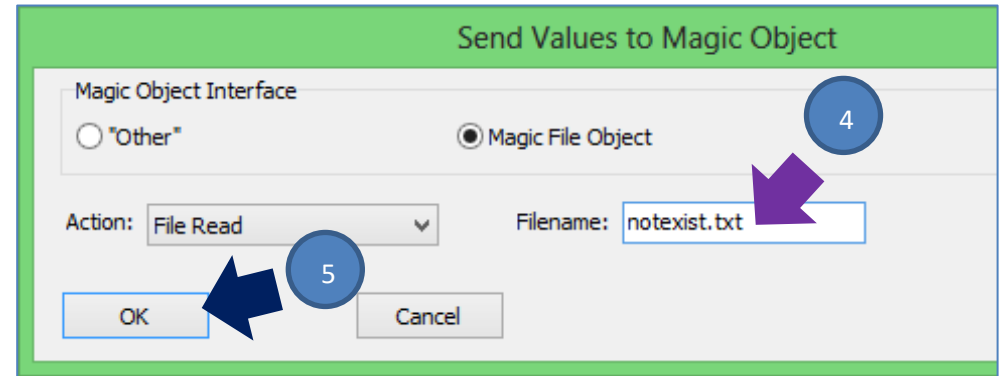
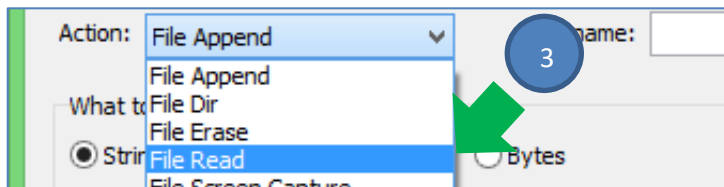
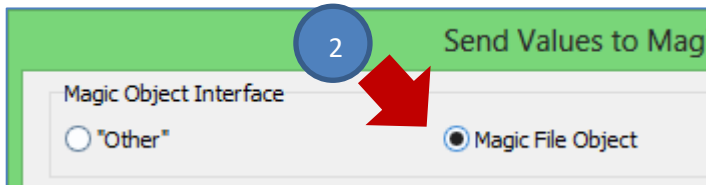
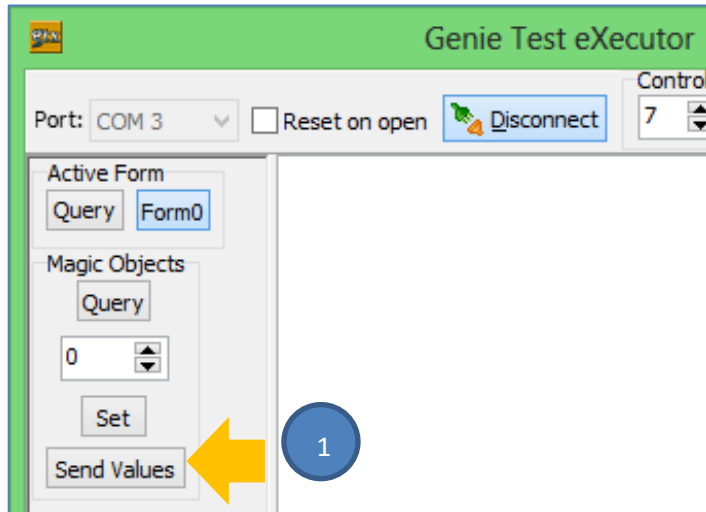
In this section we will illustrate three cases for the file read operation:

1. The file does not exist
2. The file is less than 253 bytes in size
3. The file is more than 253 bytes in size

### The File does not Exist

#### WRITE\_MAGIC\_BYTES Message

Send the MFILE\_READ command and the filename as a WRITE\_MAGIC\_BYTES message.

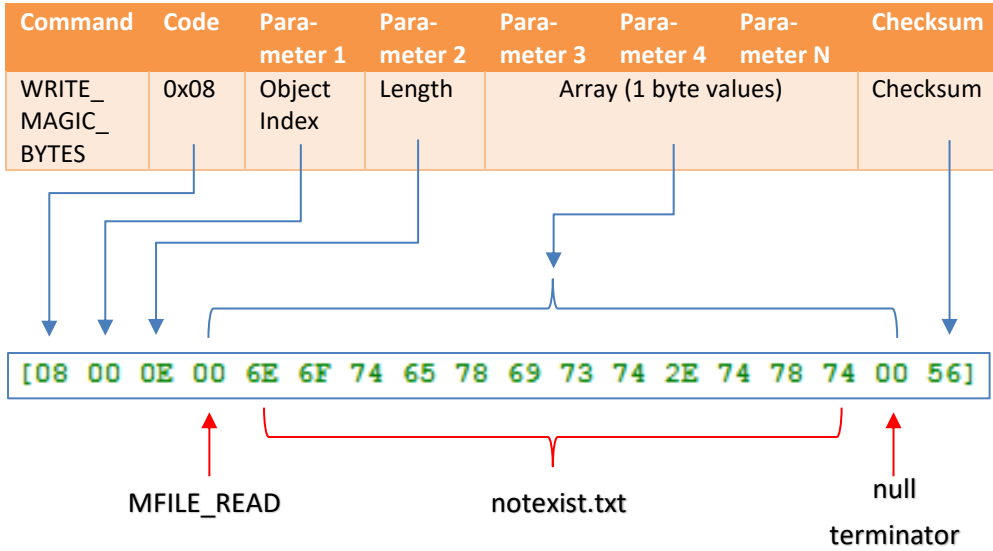


The GTX tool sends the WRITE\_MAGIC\_BYTES message.

```
Set MagicObject byte Value 10:02:47.606
```

```
[08 00 0E 00 6E 6F 74 65 78 69 73 74 2E 74 78 74 00 56]
```

The format of this message is:

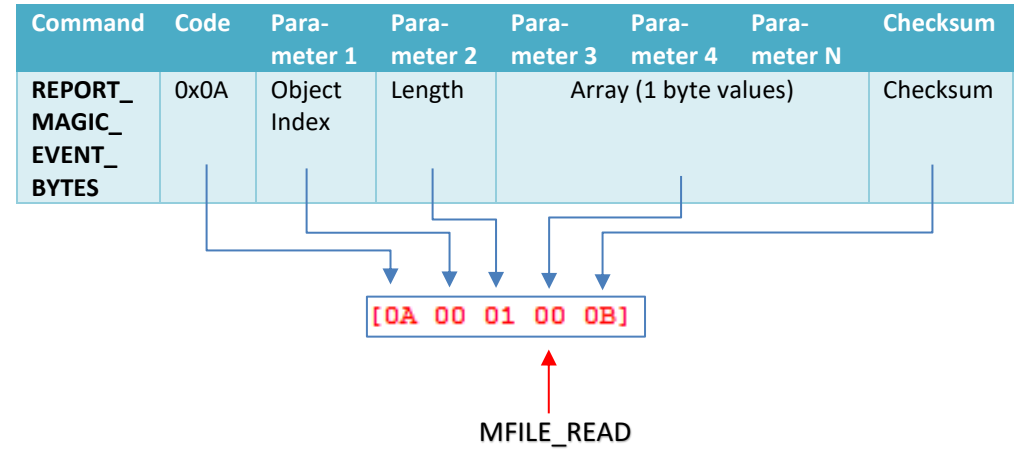


**REPORT\_MAGIC\_EVENT\_BYTES Message**

Since the file does not exist, the display module replies with a null or empty REPORT\_MAGIC\_EVENT\_BYTES message and an ACK byte.

```
Magic Object Change Bytes 10:02:47.647 [0A 00 01 00 0B]
ACK 10:02:47.648 [06]
```

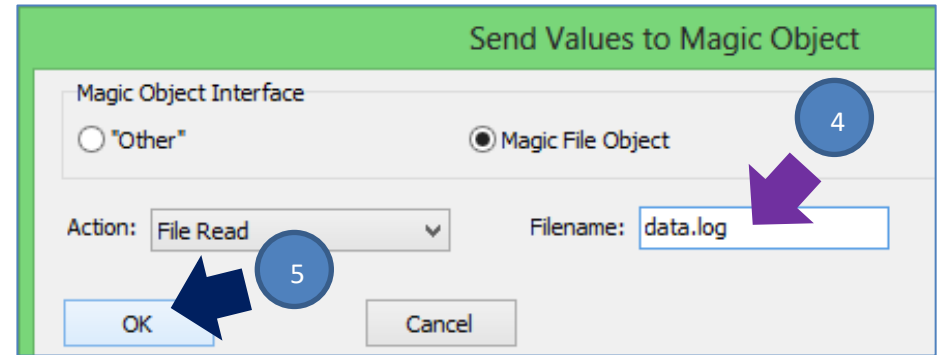
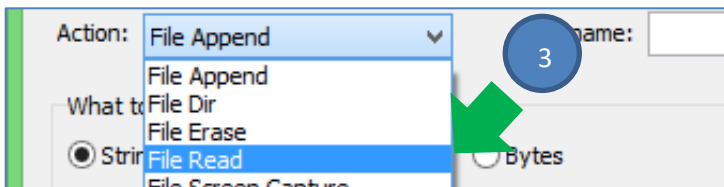
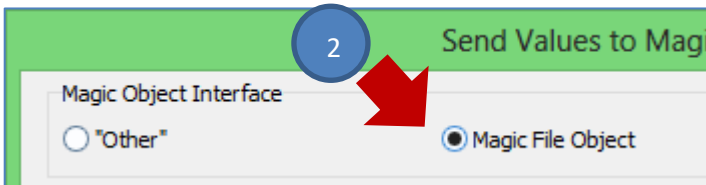
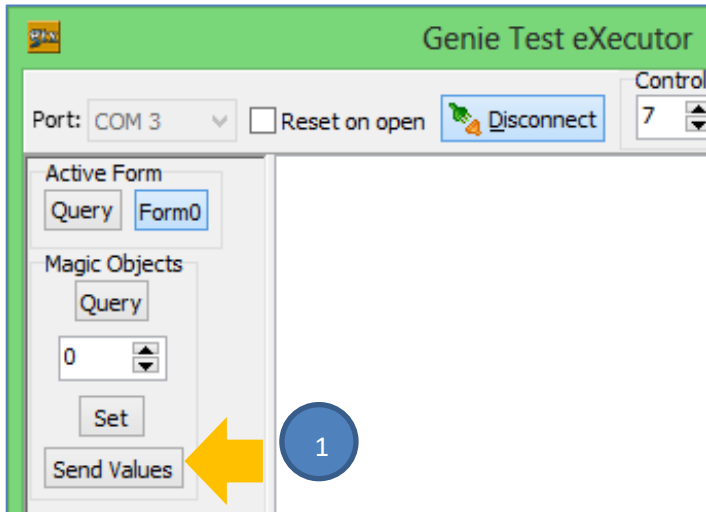
The format of this message is:



**The File is Less than 253 Bytes in Size**

**WRITE\_MAGIC\_BYTES Message**

Send the MFILE\_READ command and the filename as a WRITE\_MAGIC\_BYTES message.



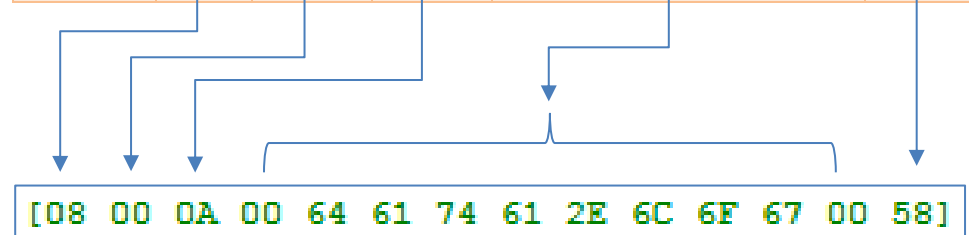
The GTX tool sends the WRITE\_MAGIC\_BYTES message.

```
Set MagicObject byte Value 15:45:20.789
```

```
[08 00 0A 00 64 61 74 61 2E 6C 6F 67 00 58]
```

The format of this message is:

Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
WRITE_MAGIC_BYTES	0x08	Object Index	Length	Array (1 byte values)			Checksum



MFILE\_READ

data.log

null terminator

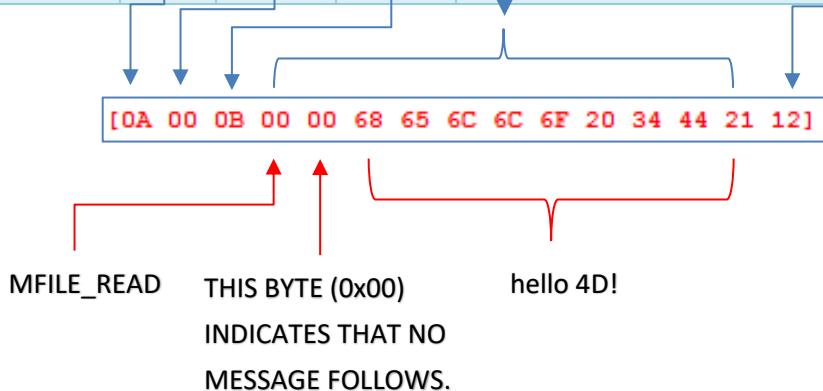
**REPORT\_MAGIC\_EVENT\_BYTES Message**

Since the file is less than 253 bytes in size, the display module replies with a single REPORT\_MAGIC\_EVENT\_BYTES message that contains the data inside the file and an ACK byte.

```
Magic Object Change Bytes 15:45:21.109
[0A 00 0B 00 00 68 65 6C 6C 6F 20 34 44 21 12]
ACK 15:45:21.113 [06]
```

The format of this message is:

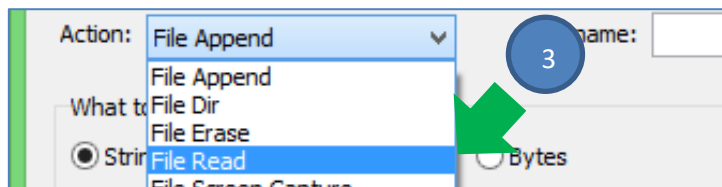
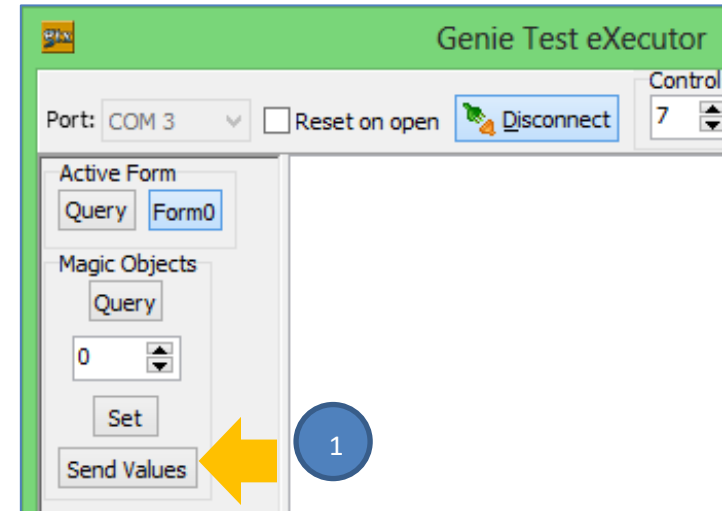
Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
REPORT_MAGIC_EVENT_BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum

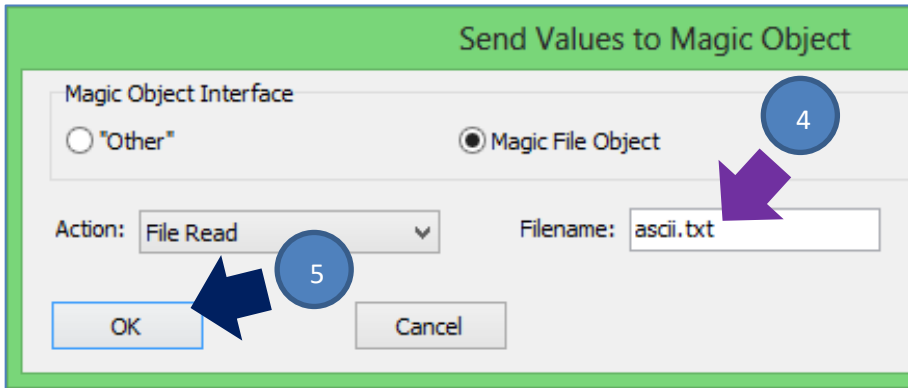


**The File is more than 253 Bytes in Size**

**WRITE\_MAGIC\_BYTES Message**

Send the MFILE\_READ command and the filename as a WRITE\_MAGIC\_BYTES message.

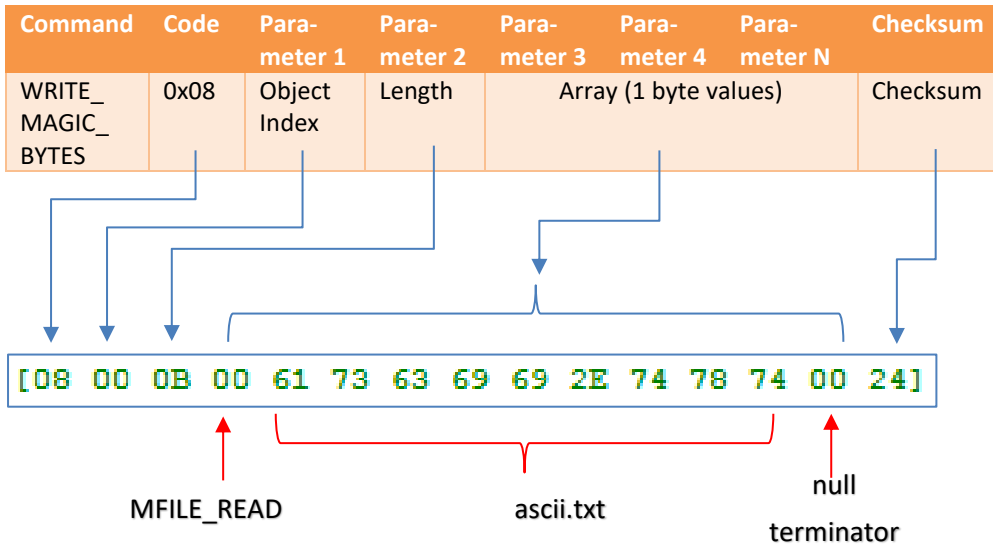




The GTX tool sends the WRITE\_MAGIC\_BYTES message.

```
Set MagicObject byte Value 16:08:00.227
[08 00 0B 00 61 73 63 69 69 2E 74 78 74 00 24]
```

The format of this message is:



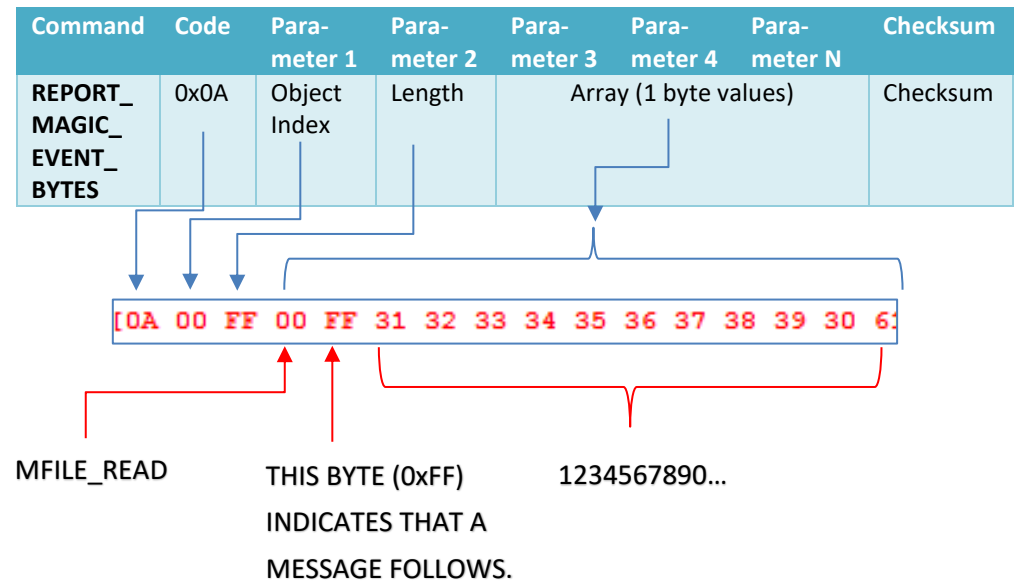
### REPORT\_MAGIC\_EVENT\_BYTES Message

Since the file is more than 253 bytes in size, the contents are divided into two packets which are then inserted into REPORT\_MAGIC\_EVENT\_BYTES messages.

```
Magic Object Change Bytes 13:20:25.314 [0A 00 FF 00 FF 31
Magic Object Change Bytes 13:20:25.376 [0A 00 43 00 00 5A
ACK 13:20:25.378 [06]
```

The format of these messages is as follows. Note that the screenshot images of the messages were cropped.

### 1<sup>st</sup> Packet



2<sup>nd</sup> Packet

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

[0A 00 43 00 00 5A 0D 0A 31 32 33 34 35 36 37 3]

MFILE\_READ

THIS BYTE (0x00)  
INDICATES THAT NO  
MESSAGE FOLLOWS.

Z..1234567...

## 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.