APPLICATION NOTES

4D SYSTEMS

TURNING TECHNOLOGY INTO ART

ViSi-Genie Magic How to Append to a File

DOCUMENT DATE: 25th APRIL 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

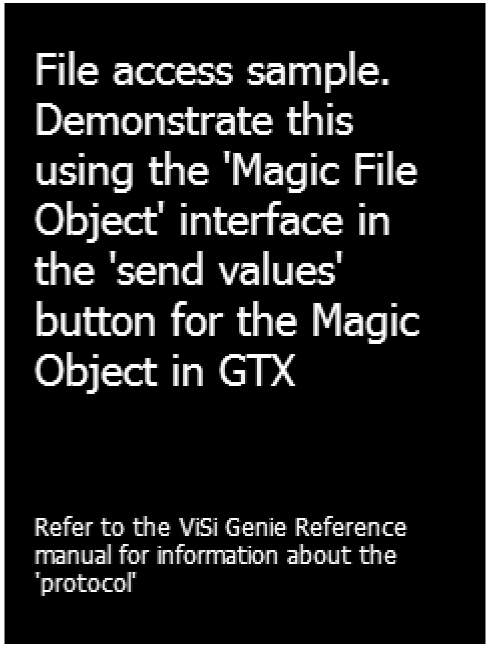
For this application note, the file access operation "MFILE_APPEND" is discussed. The implementation of a file append operation further requires the use of the following 4DGL features and functions in combination with the **Magic Object**:

- **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.



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'

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.

This application note requires:

- 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and other superseded modules which support the ViSi Genie environment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)
- Any Arduino board with a UART serial port
- 4D Arduino Adaptor Shield (optional) or connecting wires
- Arduino IDE
- 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

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 20 bytes from the host. The 20 bytes can contain an instruction byte (e.g. a file append), a null-terminated 8.3 format filename (e.g. "datalog1.txt"), and several bytes to be appended to the target file. Upon receiving these the display module will open the file (if it exists) and will append to it the received bytes. 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

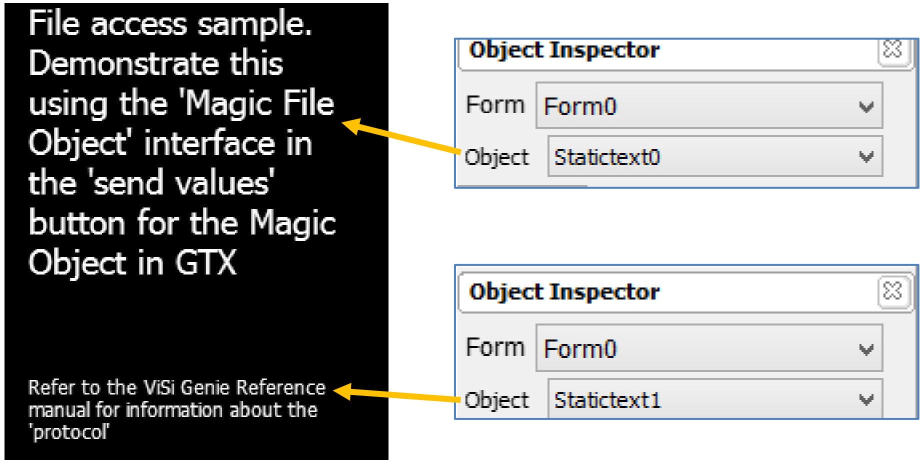
Create a New Project

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**.



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'

Object Inspector

Form: Form0

Object: Statictext0

Object Inspector

Form: Form0

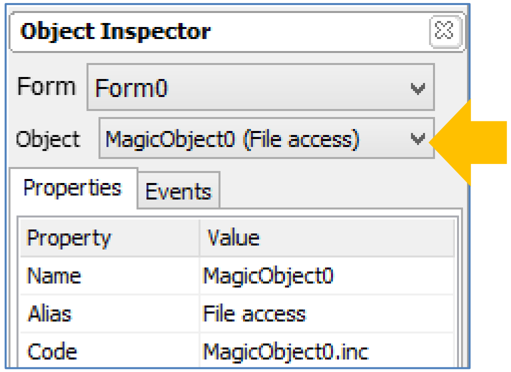
Object: Statictext1

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**.



Object Inspector

Form: Form0

Object: MagicObject0 (File access)

Property	Value
Name	MagicObject0
Alias	File access
Code	MagicObject0.inc

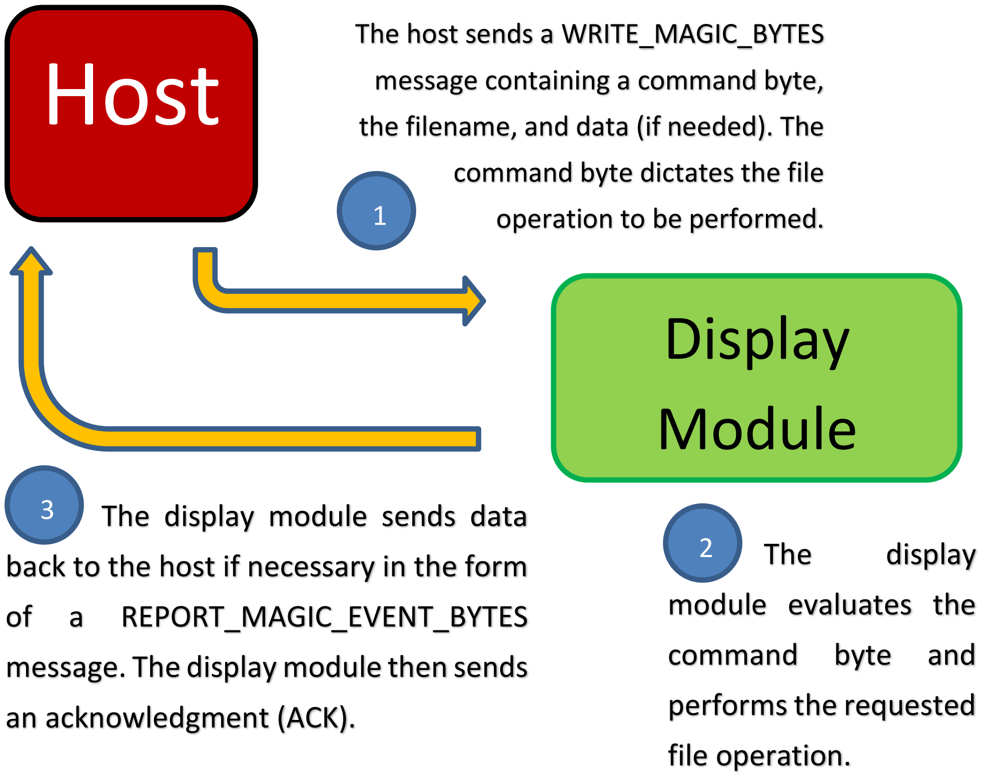
To know more about Magic Objects, their properties, and how they are added to a project, refer to the application note

[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 host sends a WRITE_MAGIC_BYTES message containing a command byte, the filename, and data (if needed). The command byte dictates the file operation to be performed.

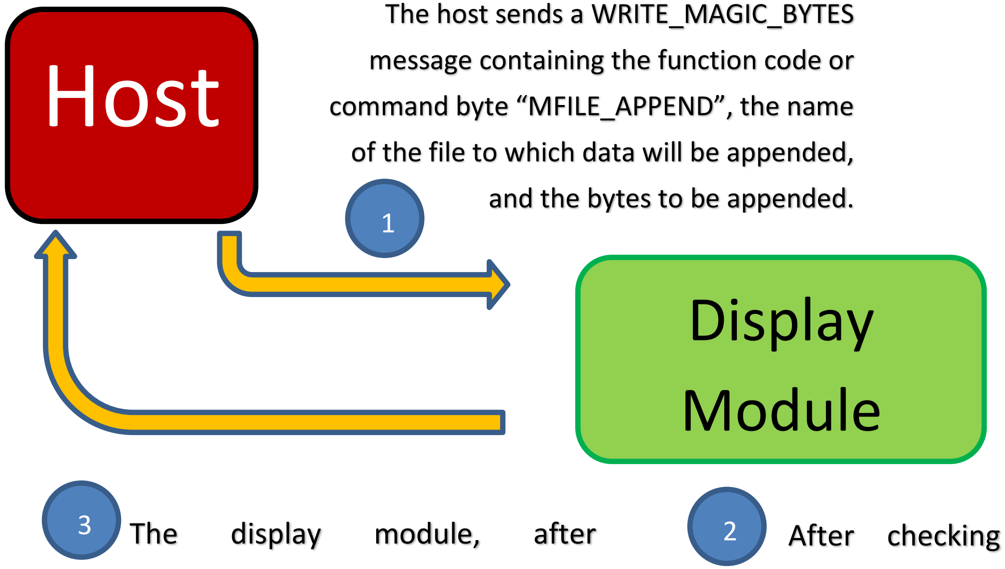
The display module evaluates the command byte and performs the requested file operation.

The display module sends data back to the host if necessary in the form of a REPORT_MAGIC_EVENT_BYTES message. The display module then sends an acknowledgment (ACK).

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 Append

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



The host sends a WRITE_MAGIC_BYTES message containing the function code or command byte “MFILE_APPEND”, the name of the file to which data will be appended, and the bytes to be appended.

The display module, after performing the file append operation, sends a REPORT_MAGIC_EVENT_BYTES message back to the host. This message contains a byte that indicates if the operation was successful or not. The display module then sends an acknowledgment (ACK).

After checking that the command byte is “MFILE_APPEND”, the display module opens the file and appends to it the received data.

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_APPEND**.

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 append operation (as expected by **FileAcces.4DGenie**).

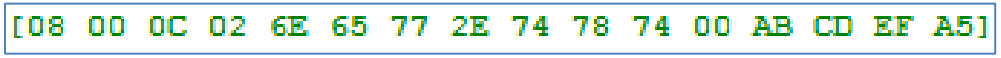
Function	Byte Value	Description and notes	Parameters	Response
MFILE_APPEND	2	Append to a file. The file must exist. A maximum of xxx bytes can be written at once.	Function code, Filename, data	Null, or True/False

Where xxx is the value set in the Workshop options configuration interface.

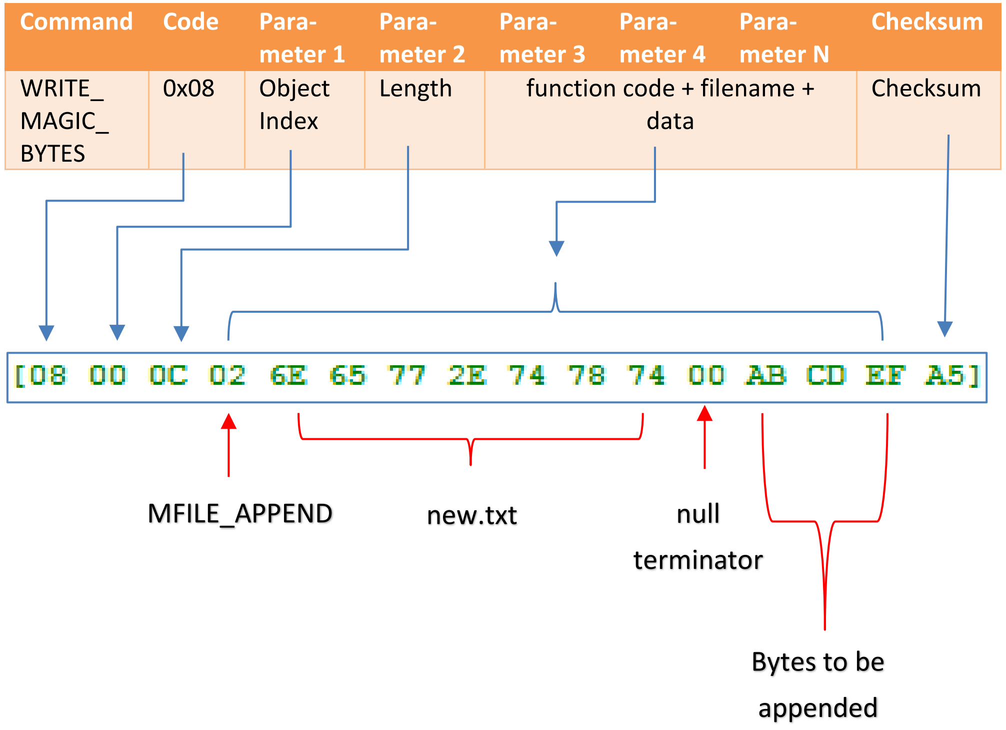
Thus, a WRITE_MAGIC_BYTES message specific to a file append operation, has the format:

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 + data			Checksum

To append to the file "**new.txt**" the bytes "0x**AB**", "0x**CD**", and "0x**EF**", the host would send:



```
[08 00 0C 02 6E 65 77 2E 74 78 74 00 AB CD EF A5]
```



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 + data			Checksum

```
[08 00 0C 02 6E 65 77 2E 74 78 74 00 AB CD EF A5]
```

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:

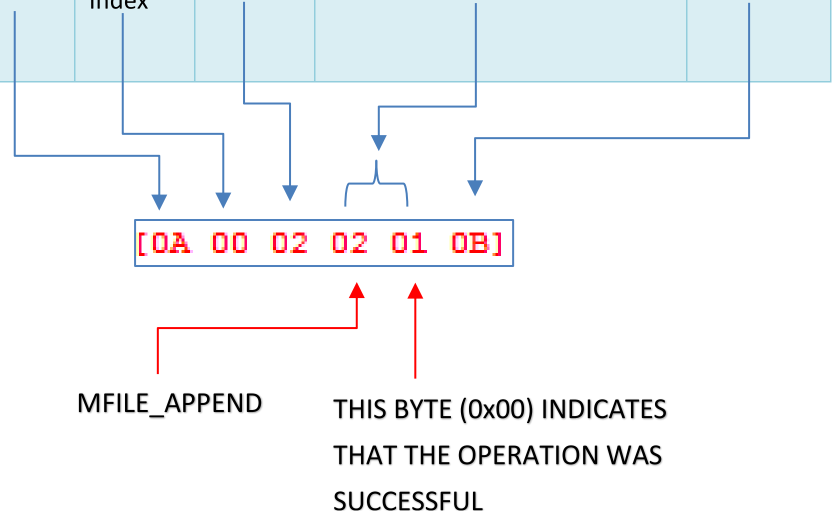
Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
REPORT_ MAGIC_ EVENT_ BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum

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 write or append operation, as implemented in **FileAcces.4DGenie**, will have the following additional data.

Command(s)	Type	Notes
Append	Function code	MFILE_WRITE or MFILE_APPEND
	'Null'	Only the 'function code' will be sent if file open fails
	1 byte	A single byte will be sent indicating the Success (true, 1) or failure (false, 0) of the operation.

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

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


[0A 00 02 02 01 0B]
MFILE_APPEND
THIS BYTE (0x00) INDICATES THAT THE OPERATION WAS SUCCESSFUL

File Open Error

If an error occurs during the file open-and-append 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 append.

The Magic Object

Going back to our working model for a file append 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 append operation (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.

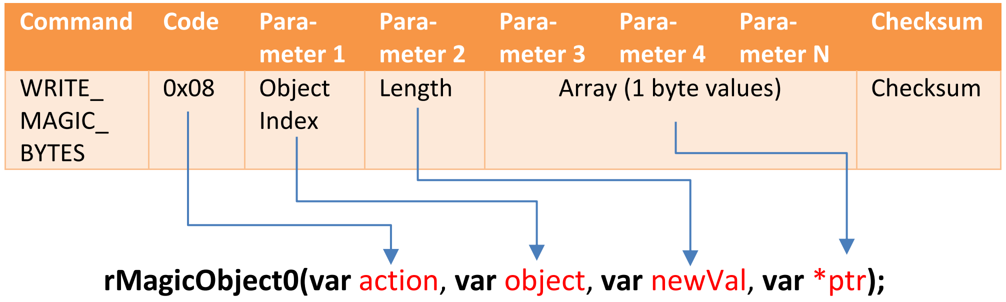
The prototype of the 4DGL function inside a **Magic Object** is:

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, the filename string, and the bytes to be appended to the target file. 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. 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 append 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”:

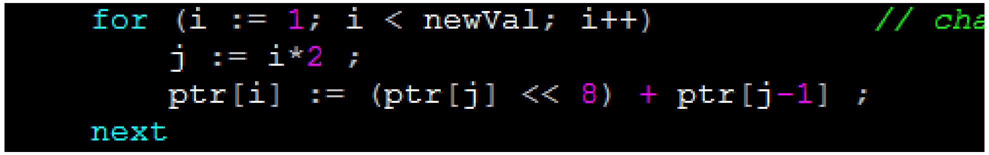
16-bit element array

address	ptr[1]	ptr[2]	ptr[3]	ptr[4]	ptr[5]	ptr[6]	ptr[7]	ptr[8]	ptr[9]	ptr[10]
Content	0x0061	0x0073	0x0063	0x0069	0x0069	0x002E	0x0074	0x0078	0x0074	0x0000
char	a	s	c	i	i	.	t	x	t	null

4DGL string

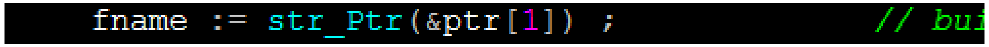
address	ptr[1]	ptr[2]	ptr[3]	ptr[4]	ptr[5]
Content	0x7361	0x6963	0x2E69	0x7874	0x0074
char	sa	ic	.i	xt	nullt

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.

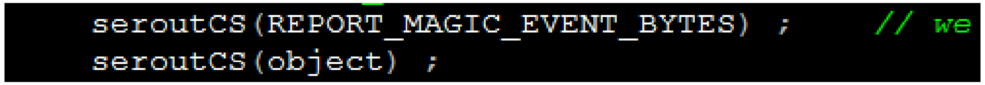


```
    fname := str_Ptr(&ptr[1]) ;                  // bui
```

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

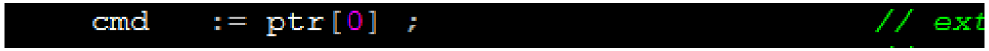
Start Constructing the REPORT_MAGIC_EVENT_BYTES Message



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

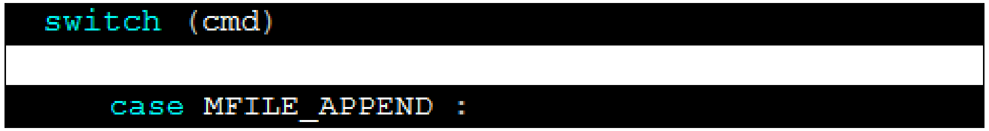
To know more about the function **“seroutCS(…)”**,see **section 5.1 Genie Magic callable Functions** of the ViSi-Genie Reference Manual.

Extract the Command Byte



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

Is cmd Equal to “MFILE_APPEND”?



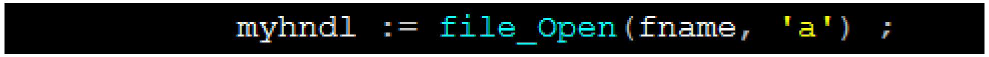
```
  switch (cmd)

     case MFILE_APPEND :
```

Diagram C. File Append Operation

The third diagram of the PDF file “**programFlow.pdf**” presents a more detailed view of the file append operation.

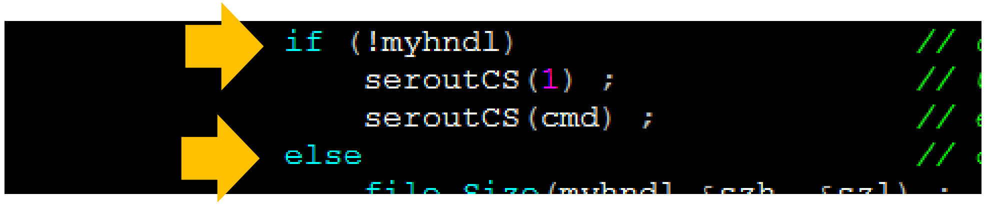
Open the File



```
            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



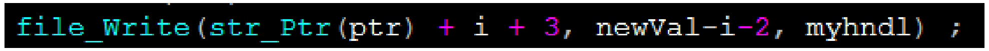
```
if (!myhndl)                    //
    seroutCS(1) ;               //
    seroutCS(cmd) ;             //
else                            //
    file_Size(myhndl, &szh, &szl) ;
```

Get the Filename String Length

```
i := str_Length(fname) ;
```

The function “**str_Length (…)**” is one of the string class functions in 4DGL. These functions are used mainly for evaluating and manipulating strings. For more information on the string class 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.

Append the Received Data to the File



```
file_Write(str_Ptr(ptr) + i + 3, newVal-i-2, myhndl) ;
```

The function “**file_Write(…)**” 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.

Identify the Messages

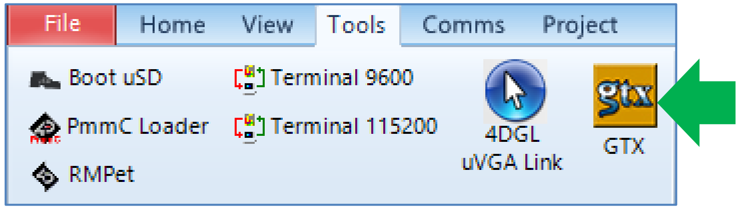
The display module is going to send and receive messages to and from an external host. This section explains to the user how to interpret these messages. An understanding of this section is necessary for users who intend to interface the display to a host. The ViSi Genie Reference Manual is recommended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

Use the GTX Tool to Analyse the Messages

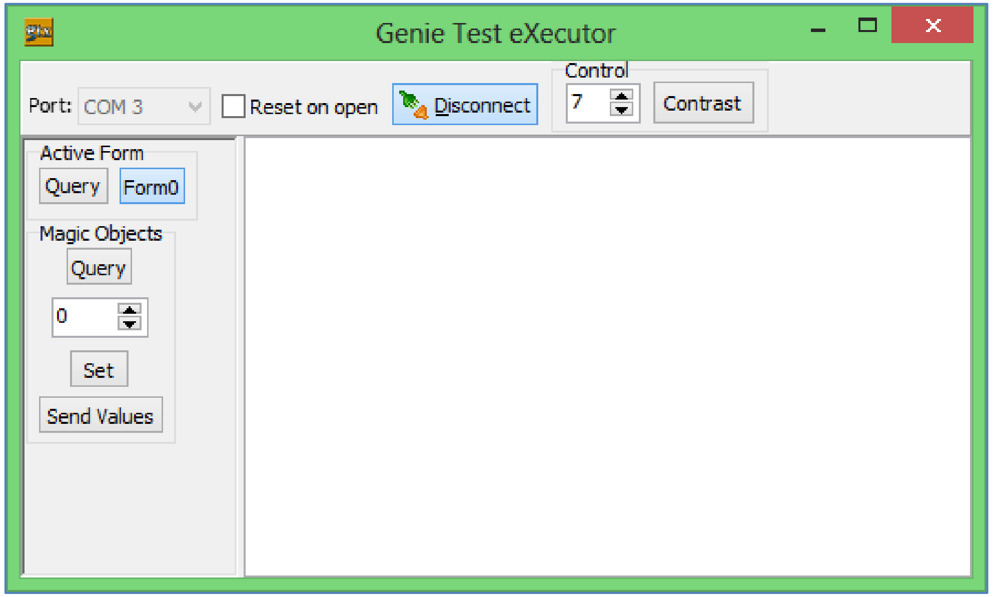
Using the GTX or **Genie Test eXecutor** tool is one option to get the messages sent by the display to the host. Here the PC will be the host. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to the display module. It is an essential debugging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

Launch the GTX Tool

Under the Tools menu click on the GTX tool button.



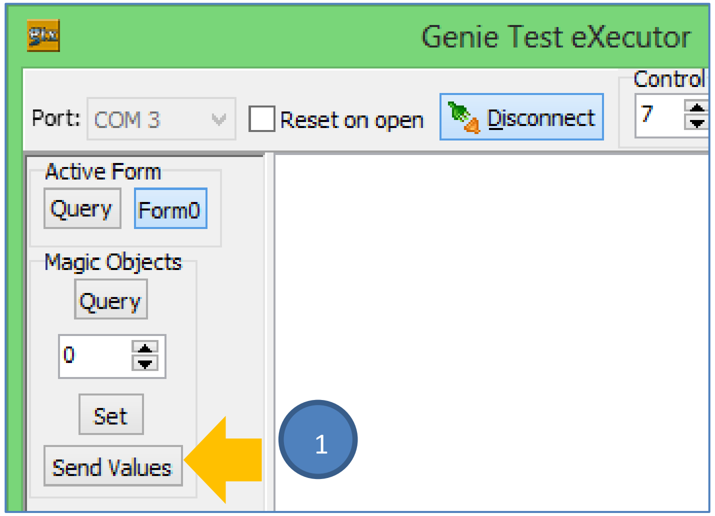
The Genie Test eXecutor window appears.



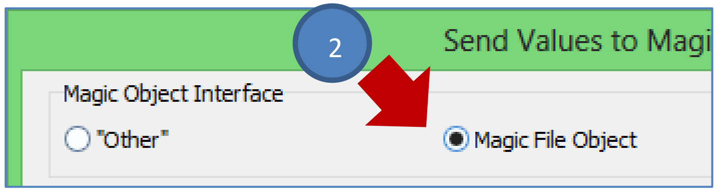
Append Hexadecimal Bytes to the Target File

WRITE_MAGIC_BYTES Message

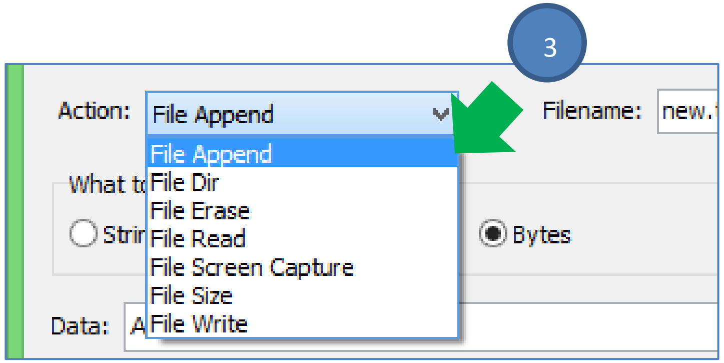
Send the MFILE_APPEND command, the filename, and the data to be appended, as a WRITE_MAGIC_BYTES message.



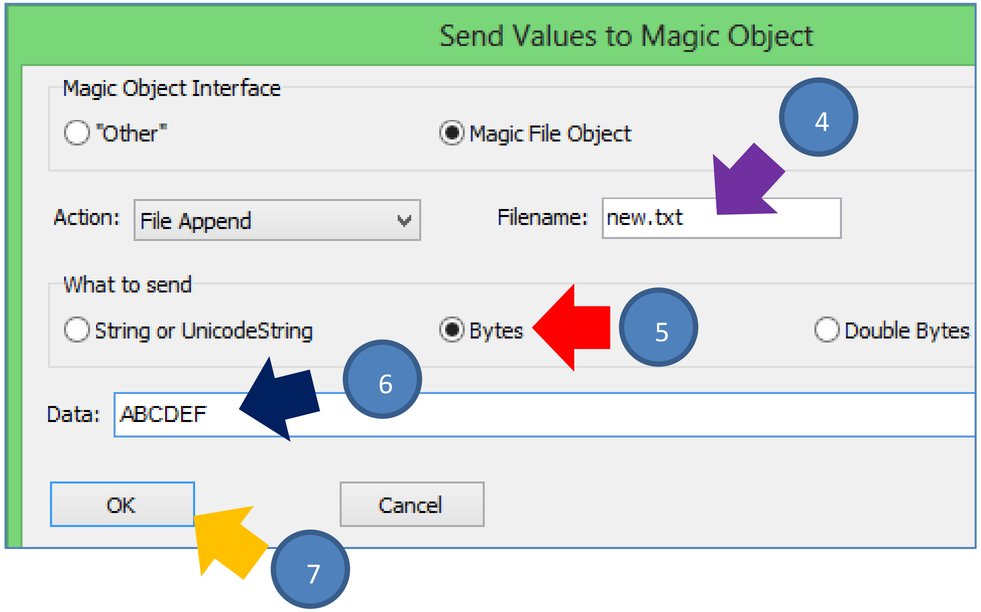
Port: COM 3
☐ Reset on open
Control: 7
Magic Objects: 0



Magic Object Interface
○ "Other" ● Magic File Object



Action: File Append
Filename: new.t
● Bytes



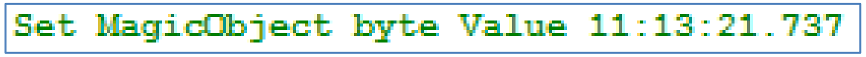
Magic Object Interface
○ "Other" ● Magic File Object

Action: File Append
Filename: new.txt

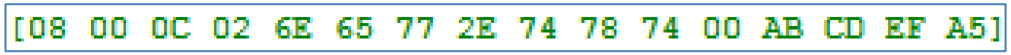
What to send
○ String or UnicodeString ● Bytes ○ Double Bytes

Data: ABCDEF

The GTX tool sends the WRITE_MAGIC_BYTES message.



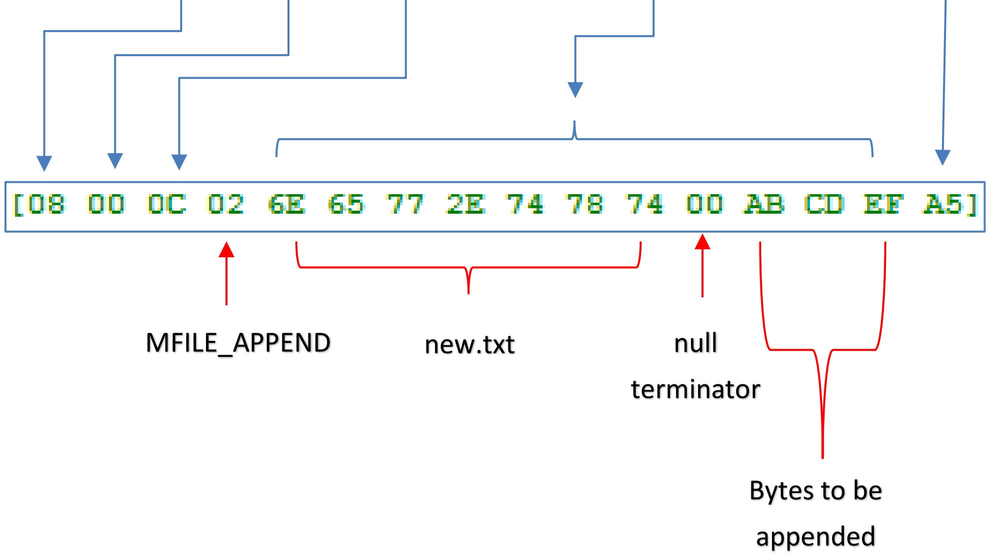
```
Set MagicObject byte Value 11:13:21.737
```



```
[08 00 0C 02 6E 65 77 2E 74 78 74 00 AB CD EF A5]
```

The format of this message is:

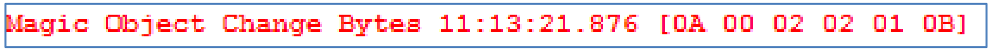
Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
WRITE_MAGIC_BYTES	0x08	Object Index	Length	function code + filename + data			Checksum



```
[08 00 0C 02 6E 65 77 2E 74 78 74 00 AB CD EF A5]
```

REPORT_MAGIC_EVENT_BYTES Message

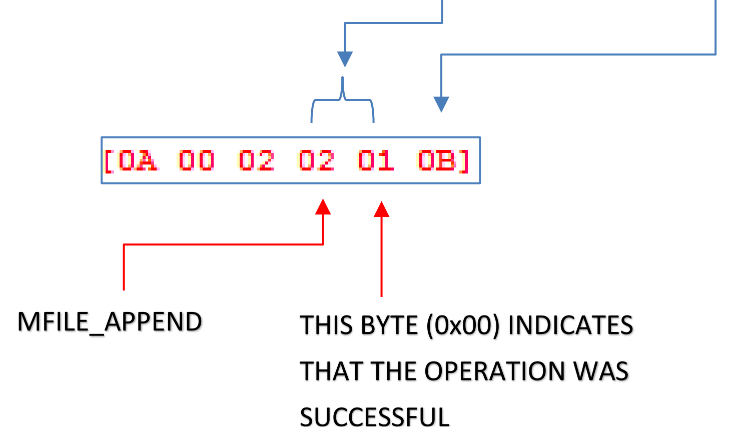
The display module replies with a REPORT_MAGIC_EVENT_BYTES message indicating that the operation was successful. Note that, as per the internal functions reference manual, the target file would actually be created if it does not exist.



```
Magic Object Change Bytes 11:13:21.876 [0A 00 02 02 01 0B]
```

The format of this message is:

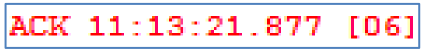
Command	Code	Para-meter 1	Para-meter 2	Para-meter 3	Para-meter 4	Para-meter N	Checksum
REPORT_MAGIC_EVENT_BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum



```
[0A 00 02 02 01 0B]
```

MFILE_APPEND
THIS BYTE (0x00) INDICATES THAT THE OPERATION WAS SUCCESSFUL

Acknowledgment Byte



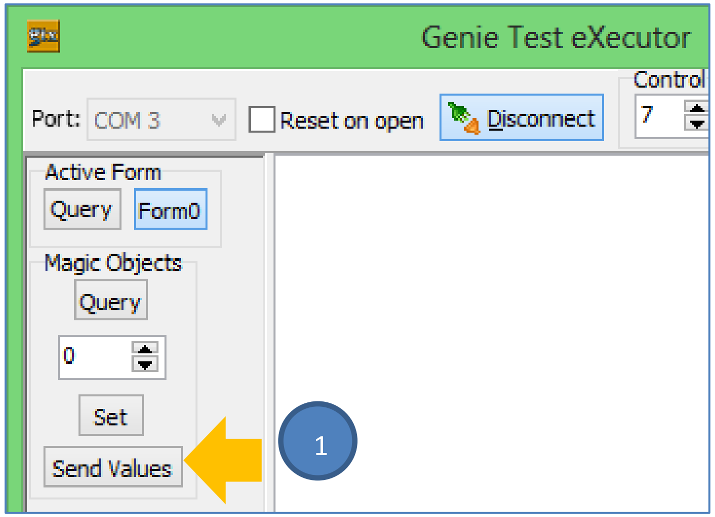
```
ACK 11:13:21.877 [06]
```

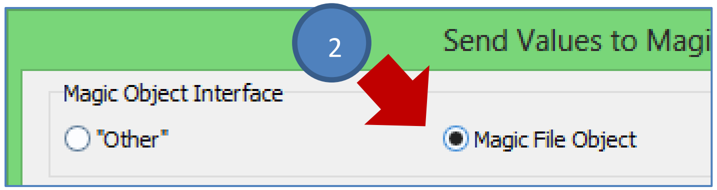
Append a String to the Target File

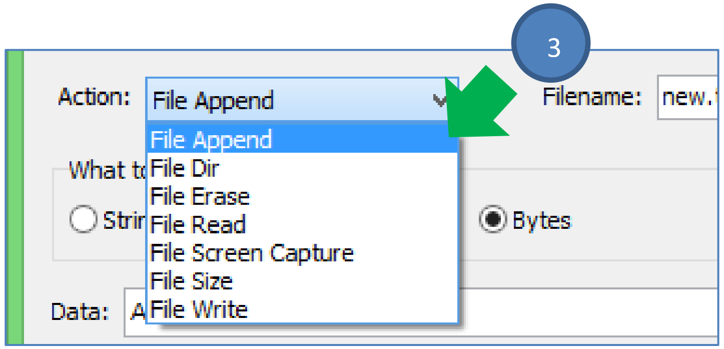
It is also possible to append a string of ASCII characters.

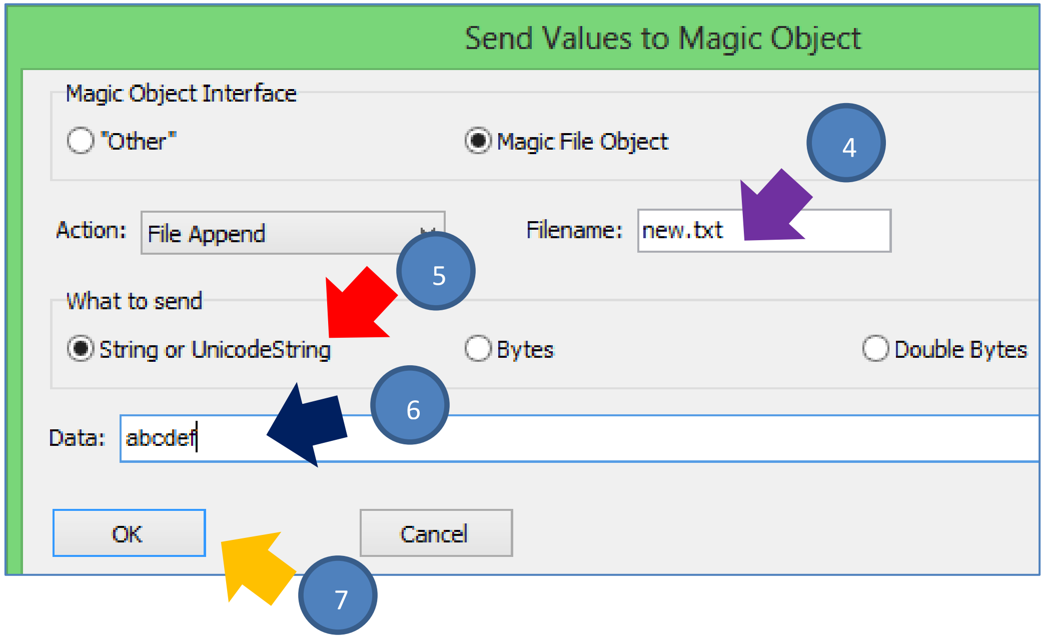
WRITE_MAGIC_BYTES Message

Send the MFILE_APPEND command and the filename as a WRITE_MAGIC_BYTES message.









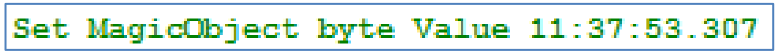
Action: File Append

Filename: new.txt

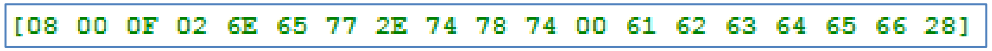
● String or UnicodeString

Data: abcdef

The GTX tool sends the WRITE_MAGIC_BYTES message.



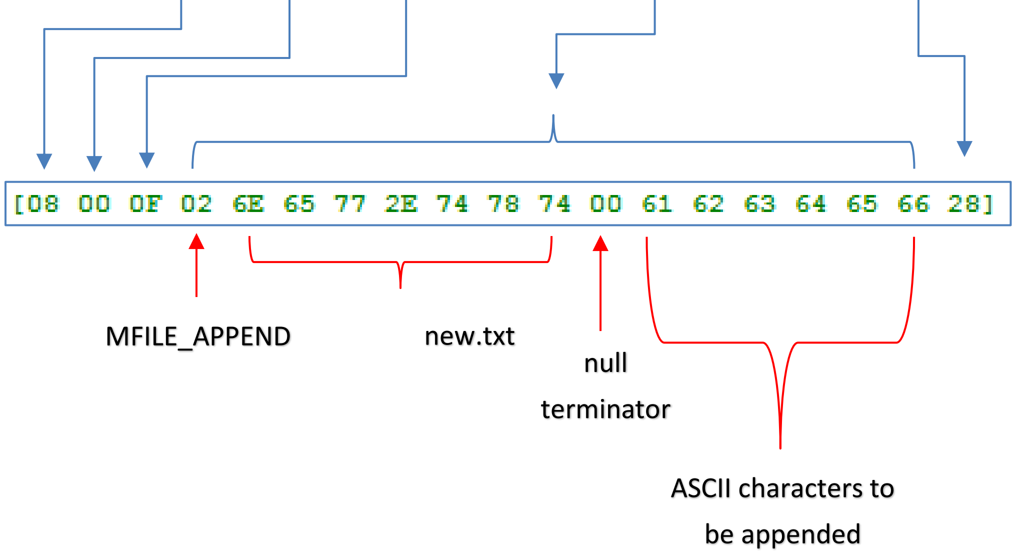
```
Set MagicObject byte Value 11:37:53.307
```



```
[08 00 0F 02 6E 65 77 2E 74 78 74 00 61 62 63 64 65 66 28]
```

The format of this message is:

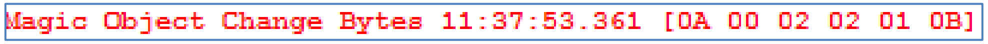
Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
WRITE_MAGIC_BYTES	0x08	Object Index	Length	function code + filename + data			Checksum



```
[08 00 0F 02 6E 65 77 2E 74 78 74 00 61 62 63 64 65 66 28]
```

REPORT_MAGIC_EVENT_BYTES Message

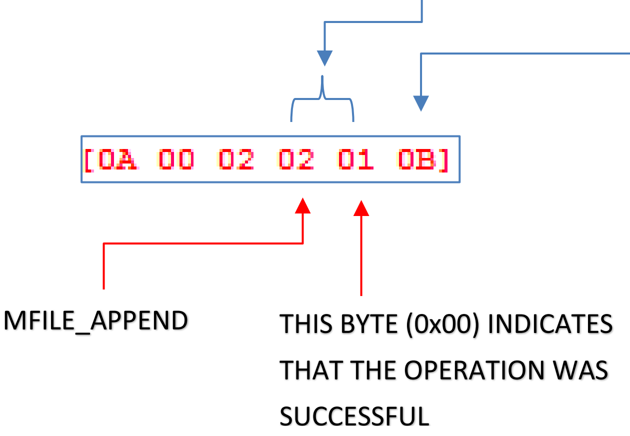
The display module replies with a REPORT_MAGIC_EVENT_BYTES message indicating that the operation was successful.



```
Magic Object Change Bytes 11:37:53.361 [0A 00 02 02 01 0B]
```

The format of this message is:

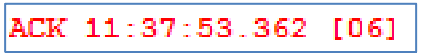
Command	Code	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter N	Checksum
REPORT_MAGIC_EVENT_BYTES	0x0A	Object Index	Length	Array (1 byte values)			Checksum



```
[0A 00 02 02 01 0B]
```

MFILE_APPEND
THIS BYTE (0x00) INDICATES THAT THE OPERATION WAS SUCCESSFUL

Acknowledgment Byte



```
ACK 11:37:53.362 [06]
```

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.