



ViSi-Genie Magic Button Counters

DOCUMENT DATE: **21st May 2019**
DOCUMENT REVISION: **1.1**



Description

This application note primarily shows how the **Magic Event** object is used to implement a project that features up and down button counters. The implementation of up and down button counters further requires the use of the following features and functions in combination with the **Magic Event** object:

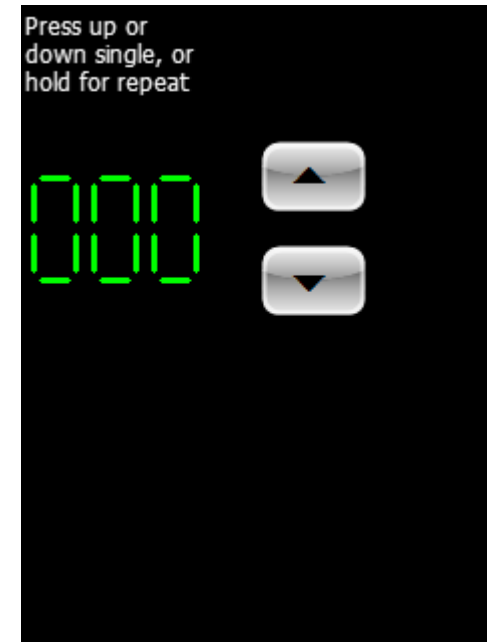
- **SendReport(...)**
- **WriteObject(...)**
- **Timers and timer events**

SendReport(...) and **WriteObject(...)** are examples of callable functions in the new ViSi-Genie Communications protocol. **SendReport(...)** causes the program to send a **REPORT_EVENT** or a **REPORT_OBJ** message to the serial port. This function can be inserted into the code of Genie Magic objects.

WriteObject(...) updates the values of Genie objects. This function can also be inserted into the code of Genie Magic objects.

Timers and **timer events** are features and functionalities in 4DGL programming. These are needed in the implementation of up and down button counters.

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



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

Note 2: Workshop Pro is needed for this application.

Before getting started, the following are required:

- Any of the following Picaso display modules:

<u>uLCD-24PTU</u>	<u>uLCD-28PTU</u>	<u>uVGA-III</u>
<u>gen4-uLCD-24PT</u>	<u>gen4-uLCD-28PT</u>	<u>gen4-uLCD-32PT</u>

and other superseded display modules which support the ViSi environment

- The target module can also be a Diablo16 display

<u>gen4-uLCD-24D</u>	<u>gen4-uLCD-28D</u>	<u>gen4-uLCD-32D</u>
<u>Series</u>	<u>Series</u>	<u>Series</u>
<u>gen4-uLCD-35D</u>	<u>gen4-uLCD-43D</u>	<u>gen4-uLCD-50D</u>
<u>Series</u>	<u>Series</u>	<u>Series</u>
<u>gen4-uLCD-70D</u>		
<u>Series</u>		
<u>uLCD-35DT</u>	<u>uLCD-43D Series</u>	<u>uLCD-70DT</u>

Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.

- [4D Programming Cable](#) / [μUSB-PA5/μUSB-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 \(μSD\)](#) 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
Design the Project.....	5
<i>Add Two Winbutton Objects to Form0.....</i>	<i>5</i>
<i>Add a LED Digits Object to Form0.....</i>	<i>6</i>
<i>Add a Static Text Object to Form0.....</i>	<i>6</i>
<i>Add Two Magic Event Objects to Form0</i>	<i>7</i>
Link a Winbutton Object to a Magic Event Object	7
<i>Timer Events.....</i>	<i>7</i>
<i>Working Model</i>	<i>8</i>
<i>Diagram A – General Program Flow.....</i>	<i>8</i>
<i>Diagram B – Internal Process.....</i>	<i>8</i>
<i>Diagram C – External Processes Model.....</i>	<i>8</i>
<i>Diagram D – External Processes Implementation</i>	<i>9</i>
Status of the Button that Triggered the Magic Event	9
Value of Leddigits0	9
Increment and Update the Value of Leddigits0	9
Send a Message to the Serial Port	9
Assign the Timer a Value	10
Attach an Event to the Timer	10

Stop the Timer	10
RepeatUp()	11
Operation	11
Build and Upload the Project.....	11
Identify the Messages	12
<i>Use the GTX Tool to Analyse the Messages.....</i>	<i>12</i>
Launch the GTX Tool	12
<i>Count Up.....</i>	<i>13</i>
REPORT_OBJ Messages	13
<i>Count Down</i>	<i>13</i>
REPORT_OBJ Messages	13
<i>REPORT_EVENT Messages</i>	<i>14</i>
Proprietary Information	16
Disclaimer of Warranties & Limitation of Liability	16

Application Overview

In the past it was not possible to create button counters in ViSi-Genie. With Workshop 4 Pro this is now possible with the use of the **Magic Event** object. The **Magic Event** object is under the **Genie Magic** pane in Workshop 4 Pro. It contains a 4DGL code and it can be linked to standard Genie objects such as a winbutton, such that any time that the button is touched, the **Magic Event** object is called (or the 4DGL code is executed). The **Magic Event** object may have a counter variable, the incrementing or decrementing value of which can be used to set the frame value of other objects (a LED digits object for instance). The value can also be sent to an external host.

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)

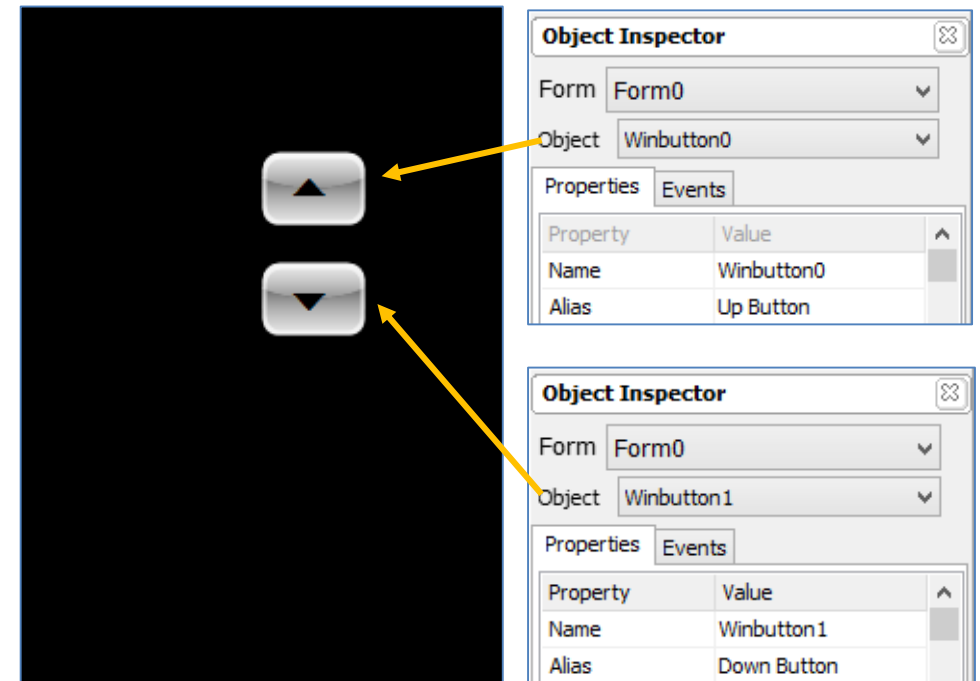
or

[ViSi Genie Getting Started – First Project for Diablo16 Displays](#) (for Diablo16).

Design the Project

Add Two Winbutton Objects to Form0

Two winbutton objects – **Winbutton0** and **Winbutton1** – are added to **Form0**.

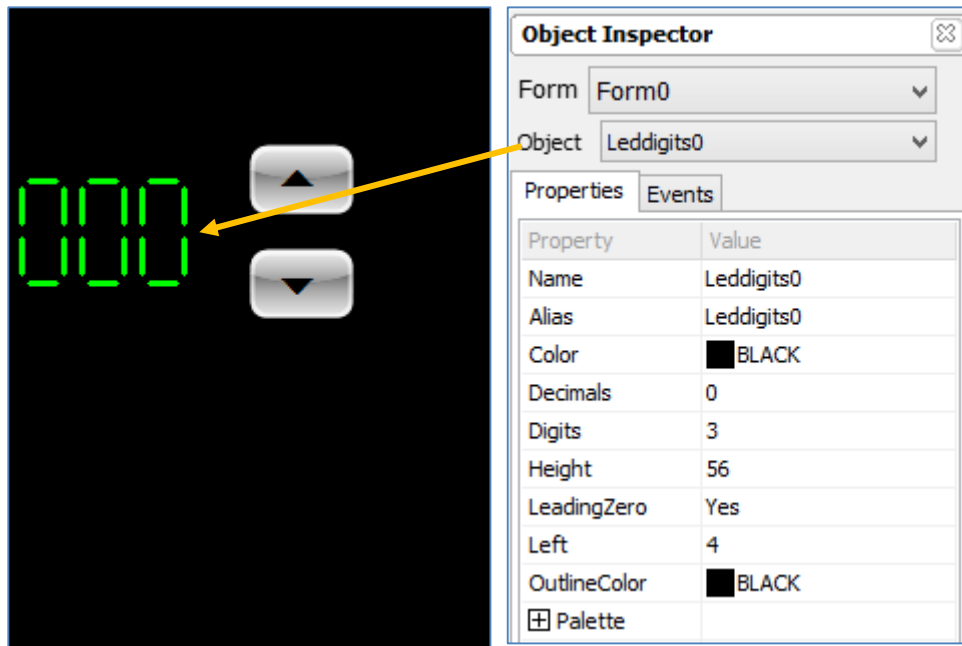


To know more about winbutton objects, their properties, and how they are added to a project, refer to the application note

[ViSi-Genie Advanced Buttons](#)

Add a LED Digits Object to Form0

A LED digits object is added to **Form0**. This is **Leddigits0**.

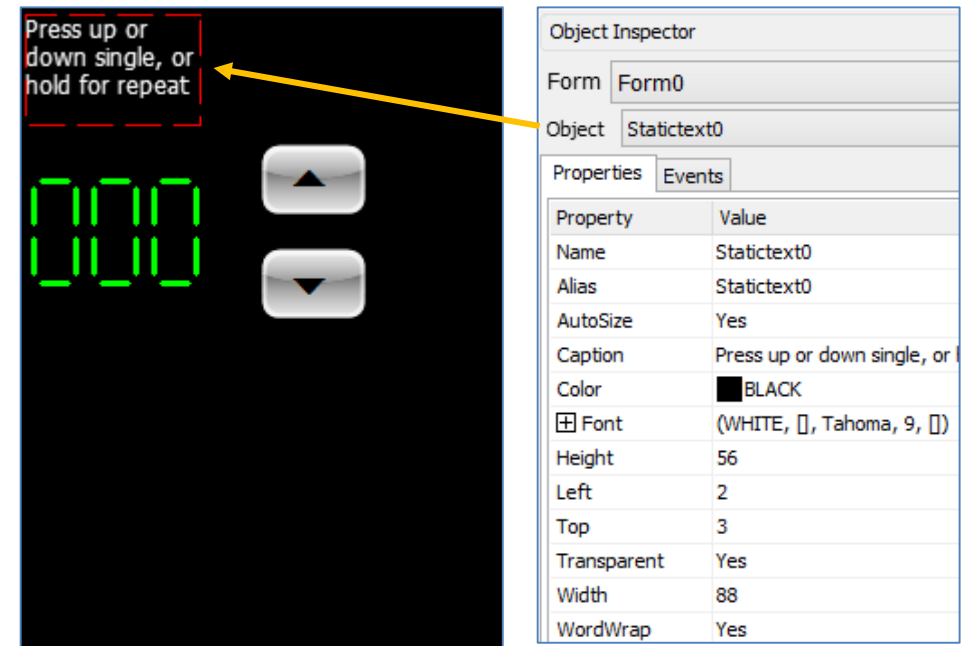


To know more about LED digits objects, their properties, and how they are added to a project, refer to the application note

[ViSi-Genie Digital Displays](#)

Add a Static Text Object to Form0

A static text object is added to Form0. This is **Statictext0**.

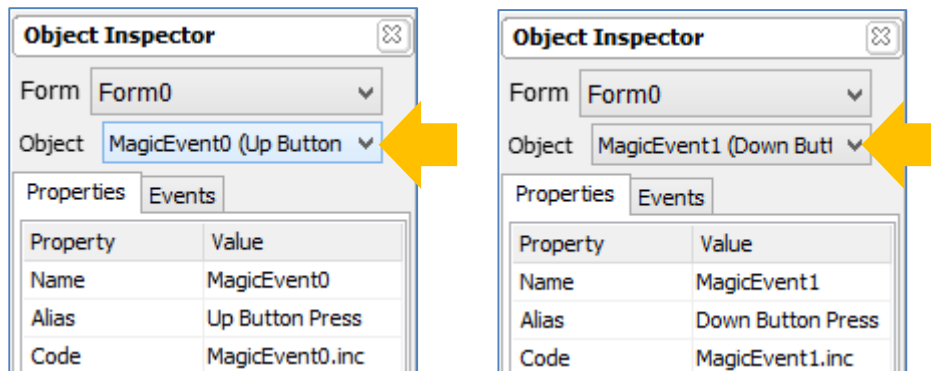


To know more about static text objects, their properties, and how they are added to a project, refer to the application note

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

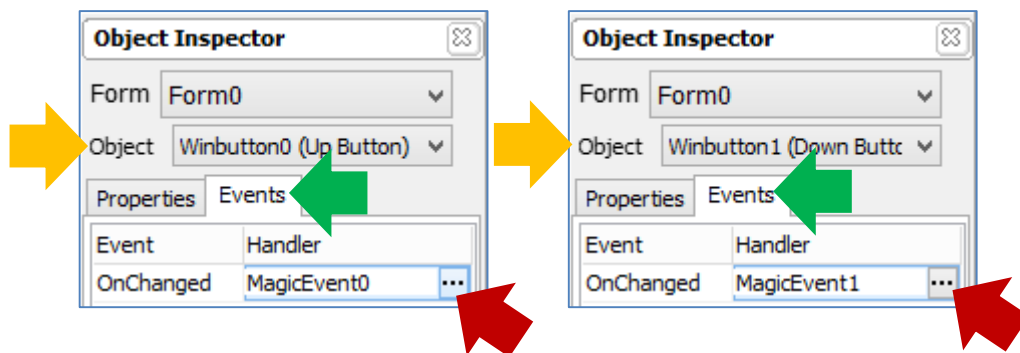
Add Two Magic Event Objects to Form0

Two magic event objects – **MagicEvent0** and **MagicEvent1** – are added to **Form0**.



Link a Winbutton Object to a Magic Event Object

Also, **Winbutton0** is linked to **MagicEvent0**, and **Winbutton1** is linked to **MagicEvent1**. Going back to the winbutton objects, the Events tabs for each are shown below.

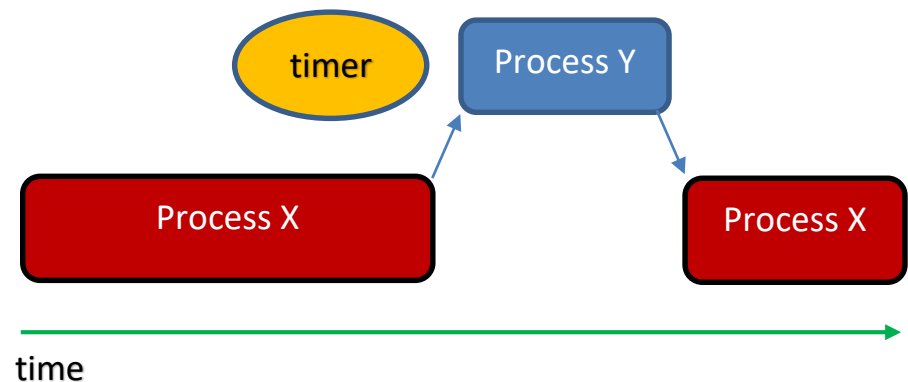


To know more about magic event objects, their properties, and how they are added to a project, refer to the application note

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

Timer Events

The Magic Event objects in the project for this application note make use of timer events. In 4DGL, a timer event is a process that is executed the moment that a timer, initially set to a certain value, reaches zero. Below is a simple illustration of how a timer event works in 4DGL programming.



Process Y above is the timer event. At some point during the execution of Process X, the timer starts to countdown. When it expires, the program counter leaves process X and goes to process Y. As soon as the program counter is done with process Y, it goes back to process X.

Here are some helpful notes related to timer events:

1. The Program Counter of the processor can only attend to one process at a time.
2. The timer counts down independently while the Program Counter is executing Process X.
3. Process Y directly depends on the timer (its value reaching 0) and not on Process X.
4. Process X or any process can initialize the value of the timer and attach or associate to it an event.
5. The timer can be stopped anytime by intentionally assigning it a value of zero. In this case, Process Y (the timer event) will not be executed. Any process can stop the timer.
6. Any process can reassign the timer a value anytime.
7. The event associated to a timer can be changed anytime during runtime.
8. In 4DGL there are 8 timers - **TIMER0** to **TIMER7** - the resolutions of which are all in milliseconds.

Working Model

Attached is a PDF file (**programFlow.pdf**) containing four diagrams that attempt to help the user understand how the demo “**UpDownRepeat.4DGenie**” works. Knowing how timer events work is the key to understanding the diagrams.

Diagram A – General Program Flow

In Diagram A there are two classifications of processes – internal and external. Internal processes are those that are performed inside Genie and are hidden from the user. External processes are those that are defined by the user through the use of the magic objects. Of course, all of the processes are actually inside Genie when the entire project is compiled. The processes are classified as such only to facilitate this discussion.

Diagram B – Internal Process

The simplified diagram is self-explanatory. Of course Genie performs other tasks besides those described in this diagram. In summary, Genie calls either **MagicEvent0** or **MagicEvent1**, depending on which button was pressed or released. Genie then passes the status of the button as an argument. Inside the magic event functions, the passed argument is “**newval**”. The variable “**newval**” therefore holds the status of the button. Also, **MagicEvent0** and **MagicEvent1** initiate the execution of other external processes.

Diagram C – External Processes Model

The right part of Diagram C is a model for the implementation of two timer events triggered, one at a time, by a timer. After comparing Diagram C to the diagram in the previous section “**Timer Events**”, the reader will see that **MagicEvent0** and **MagicEvent1** are both a **process X**, **TIMER0** is the **timer**, and **RepeatUp()** and **RepeatDown()** are both a **process Y** (timer events).

Diagram D – External Processes Implementation

Since **MagicEvent0** and **MagicEvent1** work in a similar manner, we will use only **MagicEvent0** for this discussion.

Status of the Button that Triggered the Magic Event

First, **MagicEvent0** needs to know the status of the button that caused it to be executed. We know this button to be **Winbutton0**. We also know that its status is passed as an argument. The status is passed to the variable “**newval**”. If **Winbutton0** has been pressed, a sequence of lines are executed. Otherwise, **TIMER0** is stopped and control is returned to Genie.

```
if (newval)
  i := img_GetWord(hndl, iLeddigits0, IMAGE_TAG2)
  if (i < 100)
    UpdateSend(i+1) ;
    sys_SetTimer(TIMER0, 500) ;
    sys_SetTimerEvent(TIMER0, RepeatUp) ;
  endif
else
  sys_SetTimer(TIMER0, 0) ;
endif
```

Value of Leddigits0

The current value of **Leddigits0** is stored as an entry in its image list. To access the current value of **Leddigits0**, we write

```
i := img_GetWord(hndl, iLeddigits0, IMAGE_TAG2)
```

To know more about the function “**img_GetWord(...)**”, please refer to the Picaso or Diablo16 Internal Functions Reference Manual. Right-click on the 4DGL function name text and choose “Context Sensitive help” to open the manual.

Increment and Update the Value of Leddigits0

The function “**UpdateSend(...)**” updates **Leddigits0** with a new value.

```
38 func UpdateSend(var newval)
39   WriteObject(tLeddigits, 0, newval) ;
40   SendReport(REPORT_OBJ, tLeddigits, 0,
41
42
43   endfunc
```

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

Send a Message to the Serial Port

The function “**UpdateSend(...)**” also sends a message to the serial port.


```
c UpdateSend(var newval)
  WriteObject(tLeddigits, 0, newval) ;
  SendReport(REPORT_OBJ, tLeddigits, 0, newval)
func
```

This function will cause the program to send a REPORT_OBJ message to the serial port. For more information about this function, see **section 5.1 Genie Magic callable Functions** of the [ViSi-Genie Reference Manual](#).

Assign the Timer a Value

The 4DGL function for this is

```
if (newval)
  i := img_GetWord(hndl, iLeddigits0, IMAGE_TAG2)
  if (i < 100)
    UpdateSend(i+1) ;
    sys_SetTimer(TIMER0,500);
    sys_SetTimerEvent(TIMER0,RepeatUp);
  endif
else
  sys_SetTimer(TIMER0,0);
endif
```




Here **TIMER0** is assigned a countdown period of 500 milliseconds. To know more about this function, please refer to the Picaso or Diablo16 Internal Functions Reference Manual. Right-click on the 4DGL function name text and choose “Context Sensitive help” to open the manual.

Attach an Event to the Timer

The 4DGL function for this is

```
if (newval)
  i := img_GetWord(hndl, iLeddigits0, IMAGE_TAG2)
  if (i < 100)
    UpdateSend(i+1) ;
    sys_SetTimer(TIMER0,500);
    sys_SetTimerEvent(TIMER0,RepeatUp);
  endif
else
  sys_SetTimer(TIMER0,0);
endif
```




Here **RepeatUp()** is set as the timer event for **TIMER0**. When **TIMER0** expires, **RepeatUp()** is executed. To know more about this function, please refer to the Picaso or Diablo16 Internal Functions Reference Manual. Right-click on the 4DGL function name text and choose “Context Sensitive help” to open the manual.

Stop the Timer

To stop **TIMER0** we write:

```
sys_SetTimerEvent(TIMER0,RepeatUp);
endif
else
  sys_SetTimer(TIMER0,0);
endif
```



When **TIMER0** is stopped, the timer event attached to it will not be executed.

RepeatUp()

Note that **RepeatUp()** is very similar to **MagicEvent0(...)**. Note also that **RepeatUp()** resets the value of **TIMER0** to 100 milliseconds every time.

```
func RepeatUp()  
    var i ;  
    i := img_GetWord(hndl, iLeddigits0, IMAGE_TAG2) ;  
    if (i < 100)  
        //  
        //      inputtype := tLeddigits ;  
        //      object    := oLeddigits + *(input+IPD_OBJVi  
        //      UpdateObjects(i+1) ;  
        //  
        UpdateSend(i+1) ;  
    endif  
    sys_SetTimer(TIMER0,100) ;  
endfunc
```

Operation

After analysing the above, the user should be able to conclude that when **Winbutton0** is first pressed, **MagicEvents0(...)** is called. **MagicEvents0(...)** then updates **Leddigits0** and sends a message to the serial port. After 500 ms, **RepeatUp()** is called. **RepeatUp()** also updates **Leddigits0** and sends a message to the serial port. **RepeatUp()** is then executed every after 100 ms. However, the moment that **Winbutton0** is released, **TIMER0** is stopped, and **RepeatUp()**, therefore, will not be executed. The same operation applies to **Winbutton1**, **MagicEvent1(...)**, **TIMER0**, and **RepeatDown()**.

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

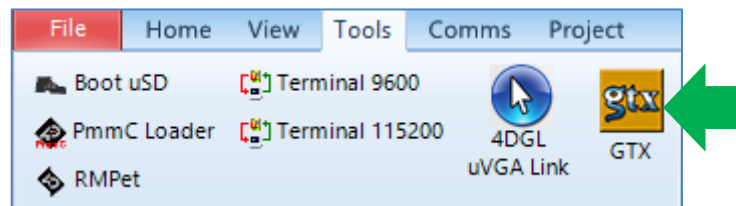
The display module is going to send messages to an external host. This section explains to the user how to interpret these messages. An understanding of this section is necessary for users who intend to interface the display to a host. The [ViSi Genie Reference Manual](#) is recommended for advanced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

Use the GTX Tool to Analyse the Messages

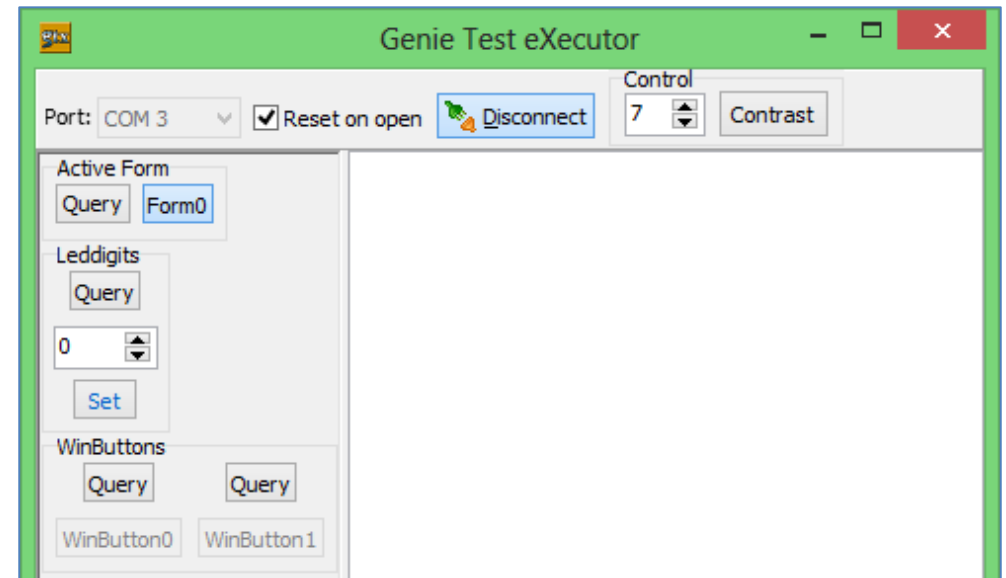
Using the GTX or **Genie Test eXecutor** tool is one option to get the messages sent by the display to the host. Here the PC will be the host. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to the display module. It is an essential debugging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

Launch the GTX Tool

Under the Tools menu click on the GTX tool button.



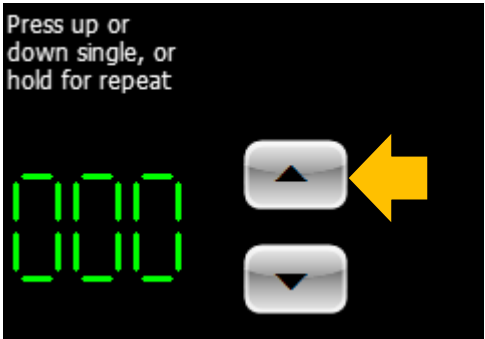
The Genie Test eXecutor window appears.



Count Up

REPORT_OBJ Messages

On the display module, press and hold the count up button.



The value of **Leddigits0** should now increment and **REPORT_OBJ** messages should be received from the display.

```
Leddigits Value 11:28:52.540 [05 0F 00 00 01 0B]
Leddigits Value 11:29:04.768 [05 0F 00 00 02 08]
Leddigits Value 11:29:05.613 [05 0F 00 00 03 09]
```

The format of the messages is shown below.

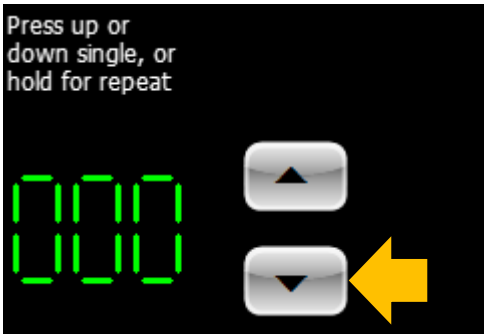
Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
0x05	0x0F	0x00	0x00	0x03	0x09
REPORT_OBJ	LED digits	First	Current value		

Upon receiving the above messages, the host would know the current value of **Leddigits0** and that that it is incrementing.

Count Down

REPORT_OBJ Messages

On the display module, press and hold the count down button.




The value of **Leddigits0** should now decrement and **REPORT_OBJ** messages should be received from the display.

```
Leddigits Value 11:29:05.613 [05 0F 00 00 03 09]
Leddigits Value 11:29:59.487 [05 0F 00 00 02 08]
Leddigits Value 11:30:02.204 [05 0F 00 00 01 0B]
Leddigits Value 11:30:03.644 [05 0F 00 00 00 0A]
```

Upon receiving the above messages, the host would know the current value of **Leddigits0** and that it is decrementing.

REPORT_EVENT Messages

Modify **MagicEvent0(...)** to make the program send **REPORT_EVENT** messages instead of **REPORT_OBJ** messages. Recompile the project and upload the program to the display module.



```
func UpdateSend(var newval)
    WriteObject(tLeddigits, 0, newval) ;
    //SendReport(REPORT_OBJ, tLeddigits, 0, newval) ;
    SendReport(REPORT_EVENT, tLeddigits, 0, newval) ;
endfunc
```

On the display module, press and hold the count up button. The value of **Leddigits0** should now increment and **REPORT_EVENT** messages should be received from the display.

```
Leddigits Change 11:31:20.803 [07 0F 00 00 01 09]
Leddigits Change 11:31:21.395 [07 0F 00 00 02 0A]
Leddigits Change 11:31:21.893 [07 0F 00 00 03 0B]
```

The format of the messages is shown below.

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
0x07	0x0F	0x00	0x00	0x03	0x0B
REPORT_EVENT	LED digits	First	Current value		

Press and hold the count down button. The value of **Leddigits0** should now decrement and **REPORT_EVENT** messages should be received from the display.

```
Leddigits Change 11:31:21.893 [07 0F 00 00 03 0B]
Leddigits Change 11:31:23.582 [07 0F 00 00 02 0A]
Leddigits Change 11:31:25.426 [07 0F 00 00 01 09]
Leddigits Change 11:31:27.118 [07 0F 00 00 00 08]
```

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.