



ViSi Genie User Images

DOCUMENT DATE: **1st MAY 2020**
DOCUMENT REVISION: **1.2**



Description

This application note provides a first hands-on example with ViSi-Genie and describes all the steps related to a project.

Before getting started, the following are required:

- Workshop 4 has been installed according to the document Workshop 4 Installation;
- The user is familiar with the Workshop 4 environment and with the fundamentals of ViSi-Genie, as described in Workshop 4 User Guide and ViSi-Genie User Guide;
- 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 discussed in these recommended application notes.

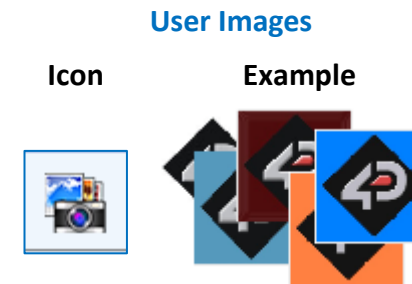
Content

Description	2
Content	2
Application Overview	3
Setup Procedure	4
Create a New Project	4
<i>Create a New Project</i>	4
Design the Project	4
<i>Adding a User Images Object</i>	5
Adding Images	6
<i>Naming of Objects</i>	7
<i>Creating a Button-controlled Slideshow</i>	8
Adding a Win Button	8
Linking Objects	9
Creating the “Next” Button	10
Configuring the User Images Object to Report a Message	11
Adding a Border Background	11
Selecting Objects	12
Testing the Slideshow	13
<i>Creating a Timer-driven Slideshow</i>	13
Add another User Images Object	13
Adding a Timer Object	14
Configuring a Timer to Drive a User Images Object	15

Add Momentary User Buttons to Control the Timer Object	15
Defining the States of a Momentary User Button	17
Link the User Buttons to Timer0	20
Add Three Win Buttons to Control UserImages1	20
Build and Upload the Project	22
Identify the Messages	23
<i>Use the GTX Tool to Analyse the Messages</i>	23
Launch the GTX Tool	23
<i>The User Images Object</i>	23
Report Event	23
Control a User Images Object using the GTX Tool	25
Polling a User Images Object	26
Proprietary Information	27
Disclaimer of Warranties & Limitation of Liability	27

Application Overview

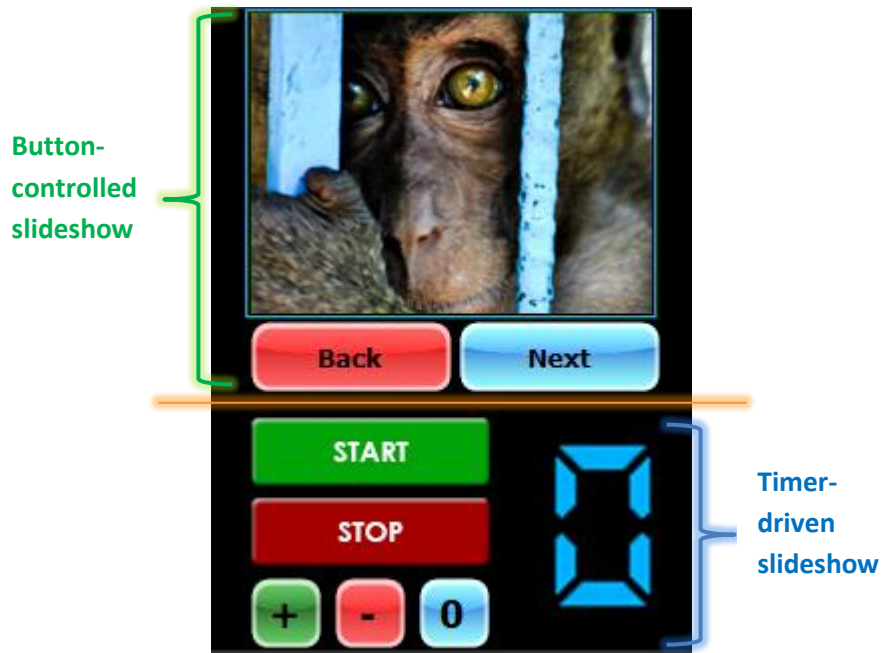
It is often difficult to design a graphical display without being able to see the immediate results of the application code. ViSi-Genie is the perfect software tool that allows users to see the instant results of their desired graphical layout with this large selection of gauges and meters (called objects or widgets). The user can simply click on the desired widget to select it and click on the simulated display to place the widget. Shown below is the user images widget.



The user images object represents an easy way to build a slideshow by joining together a sequence of images in one place. The user provides the images and can use an input type object, such as a button or a slider, to make the user images object display the next or previous frame.

The user images object can also be made to behave like a video player with the use of a timer. Each click of the timer will make the user images object increment to the next frame.

The project developed in this application note illustrates how to create an input-driven (or button-controlled) slideshow and a timer-driven (video player-like) slideshow.



The section “**Identify the Messages**” discusses the format of the messages coming from and going to a user images object using the GTX Tool in Workshop. An understanding of this section is essential for users who intend to interface the display 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).

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)

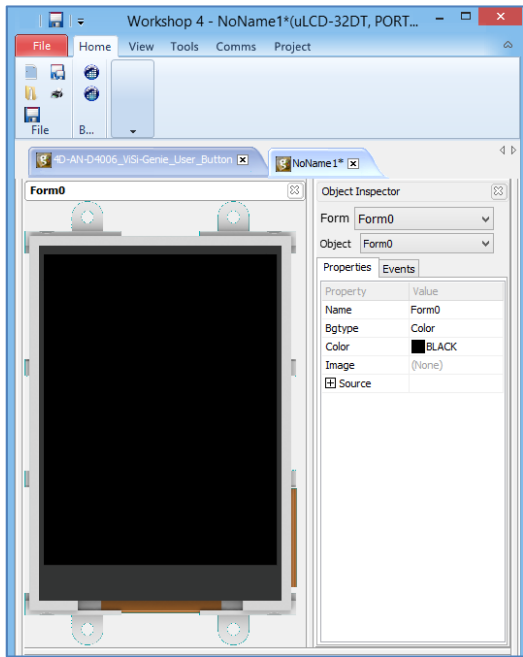
or

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

Design the Project

Everything is now ready to start designing the project. **Workshop 4** displays an empty screen, called **Form0**. A **form** is like a page on the screen. The form can contain **widgets** or **objects** like trackbars, sliders, displays or keyboards.

Below is an empty form.

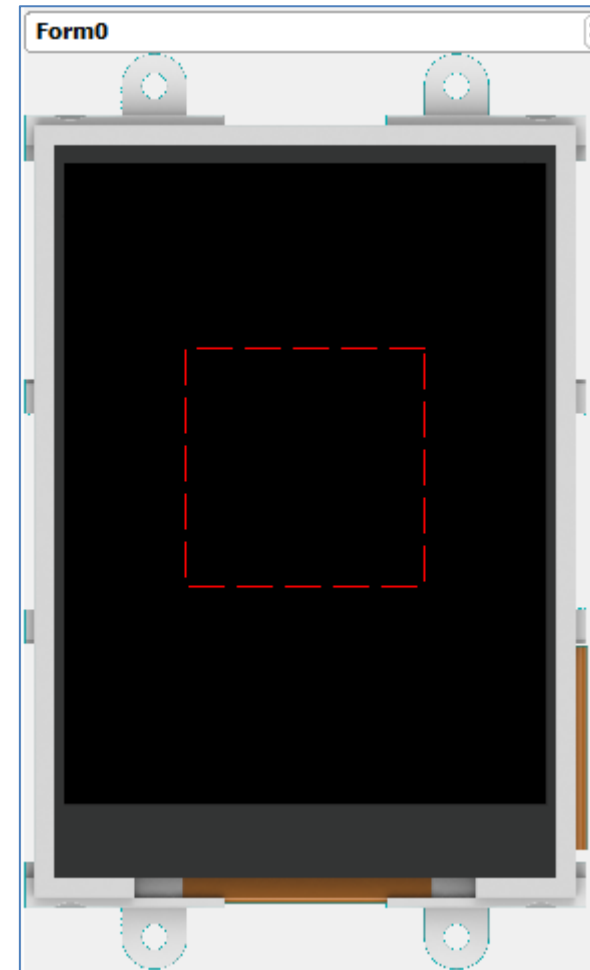


Adding a User Images Object

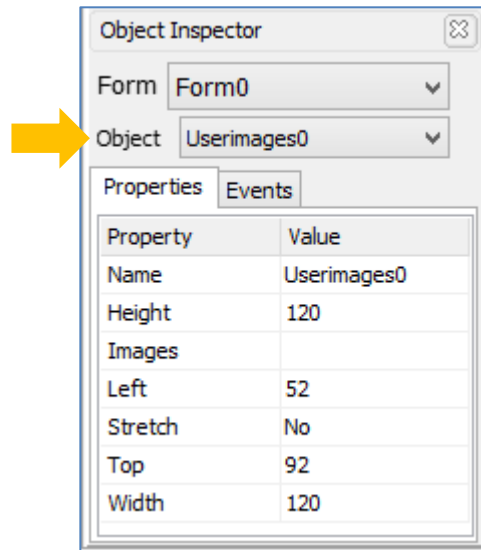
To add a user images object, go to the **Systems/Media** pane then click on the **User Images** icon.



Click on the **WYSIWYG** (What-You-See-Is-What-You-Get) screen to put the object in place. The WYSIWYG screen simulates the actual appearance of the display module screen.

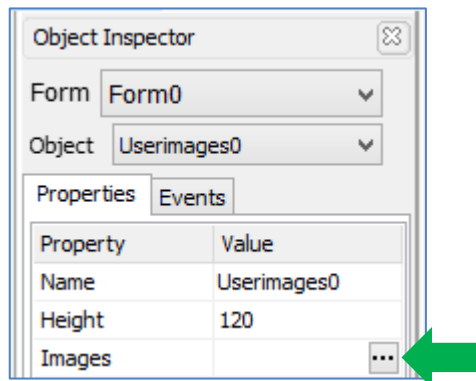


The empty user images object can be dragged to any desired location. The **Object Inspector** on the right part of the screen displays all the properties of the newly created user images object named **Userimages0**.

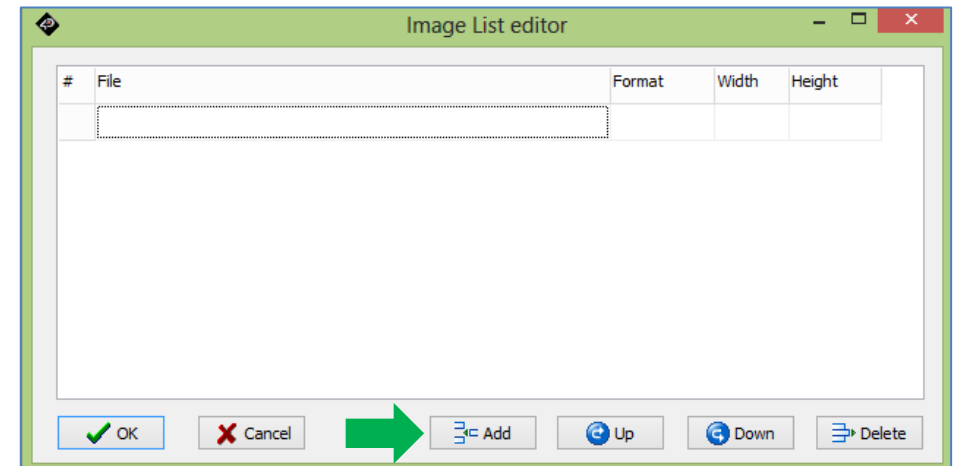


Adding Images

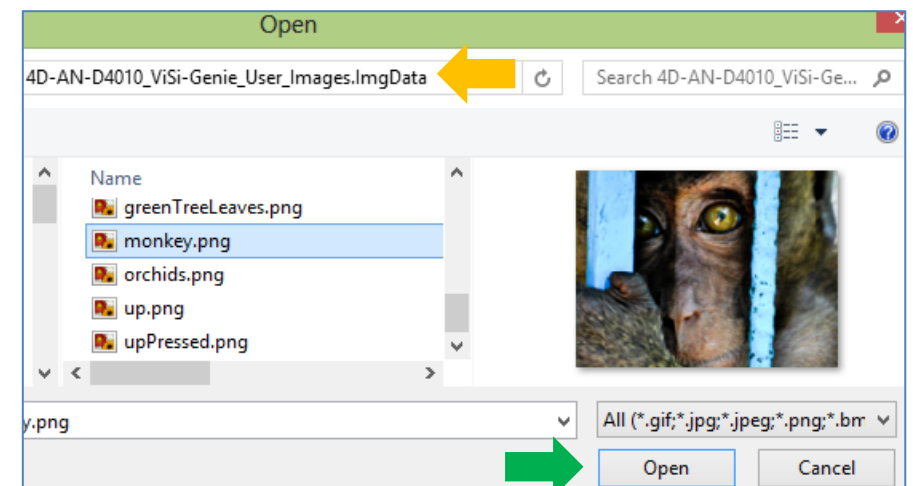
To add images to the object, click on the ellipsis dots of the property “Images” in the object inspector.



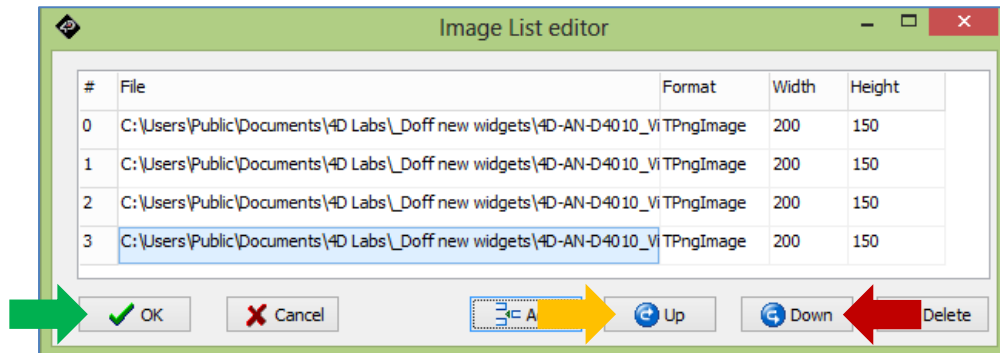
The Image List editor window appears. Click on the Add button.



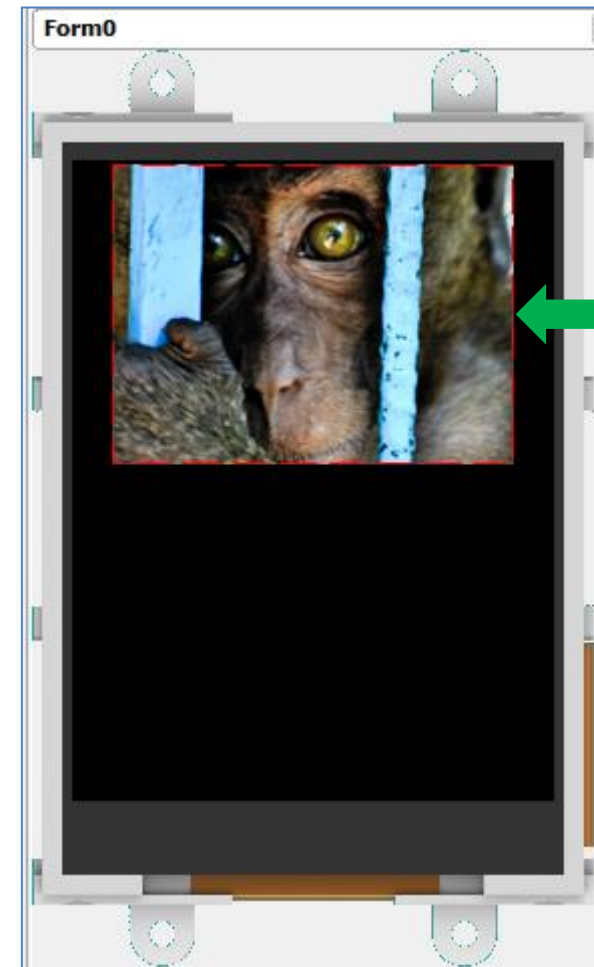
A standard Open window appears and asks for image files. The image files used in this project are in the “.ImgData” folder which is automatically generated when the demo project is compiled. Multiple image files can be selected. Click on the “Open” button when done selecting the files.



The Image List editor window reappears with a list of the images. To rearrange the order, select an image and press the Up or Down button. Click OK when done.



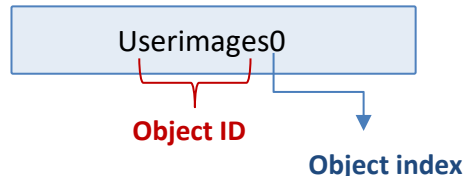
The WYSIWYG screen is updated accordingly, showing the first image on the list.



Naming of Objects

Naming is important to differentiate between objects of the same kind. For instance, suppose the user adds another user images object to the WYSIWYG screen. This object will be given the name Userimages1– it being

the second user images object in the program. The third user images object will be given the name Userimages2, and so on. An object's name therefore identifies its kind and its unique index number. It has an ID (or type) and an index.

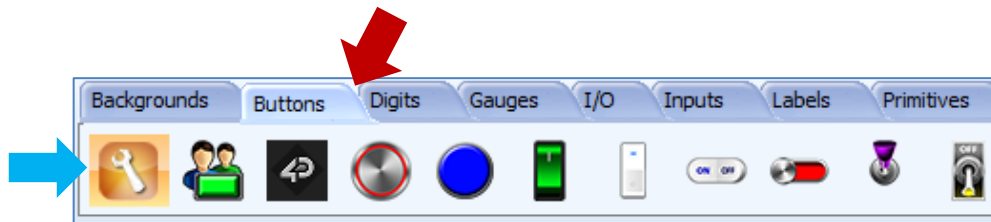


Creating a Button-controlled Slideshow

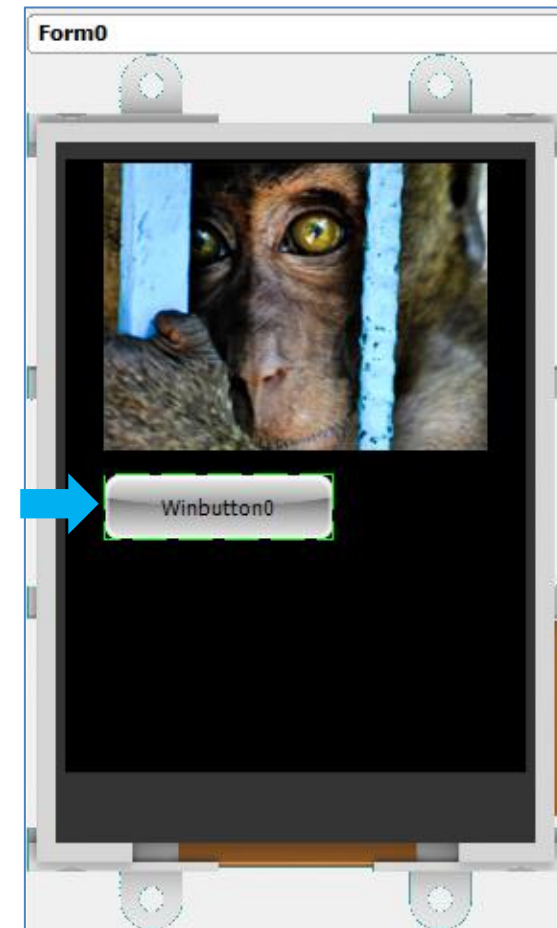
Userimages0, which contains four images, will be used to create a slideshow. Two win or fancy buttons will control the slideshow – one for displaying the next frame and one for displaying the previous frame.

Adding a Win Button

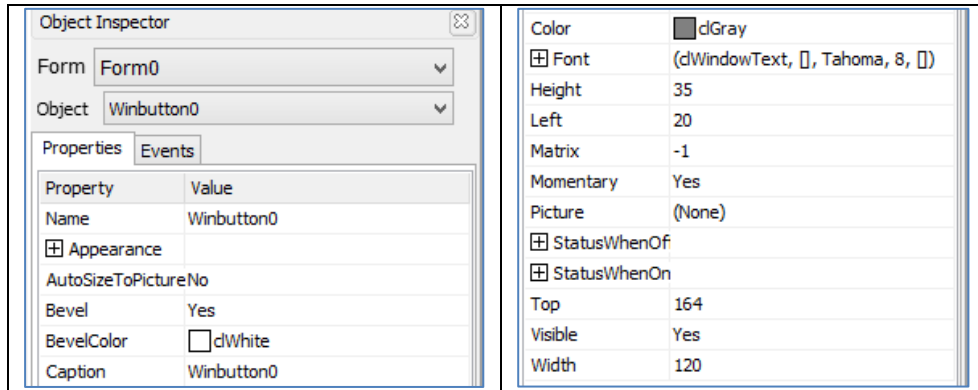
To add a win or fancy button, go to the Buttons pane and click on the win button icon.



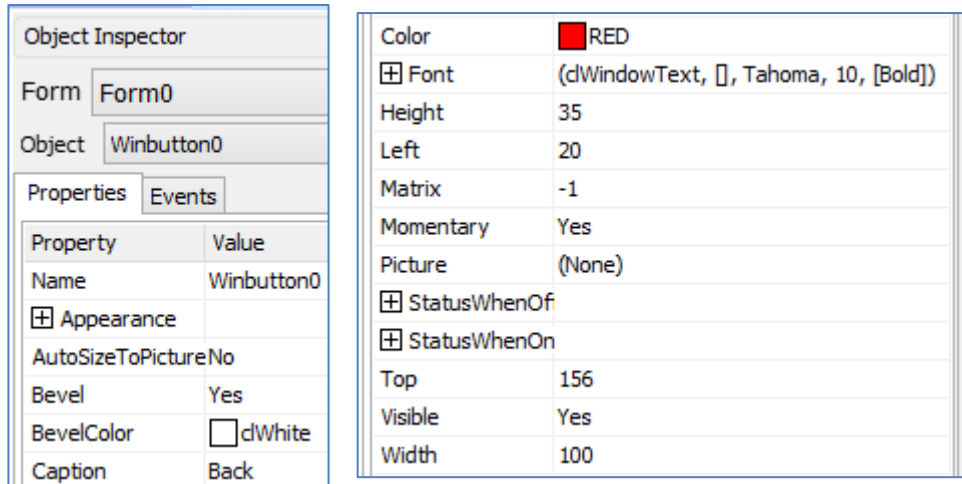
Click on the WYSIWYG screen to place the object.



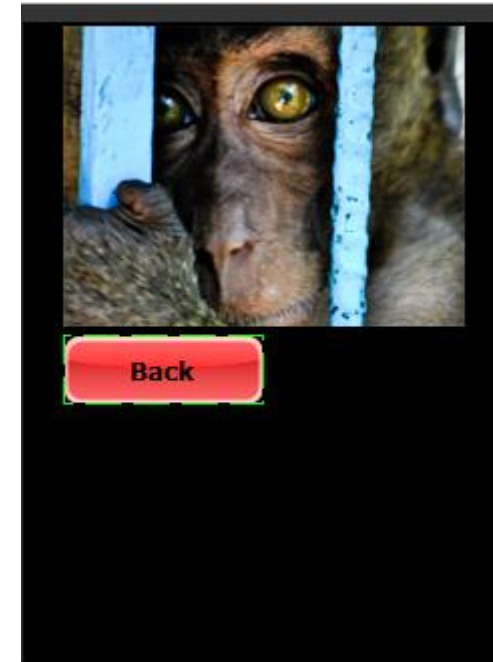
The object inspector shows all the properties of the newly-created win button, named **Winbutton0**.



Take time to experiment with the different properties of a win button. Winbutton0 of this project has the following properties.

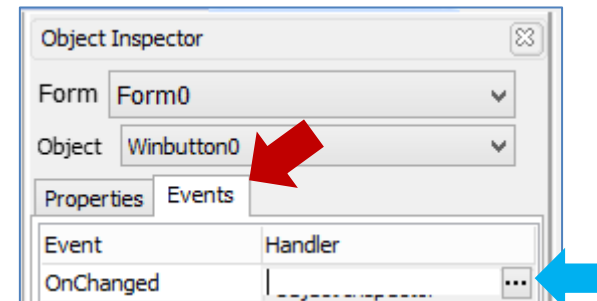


When done, the WYSIWYG screen will look similar to that shown below.

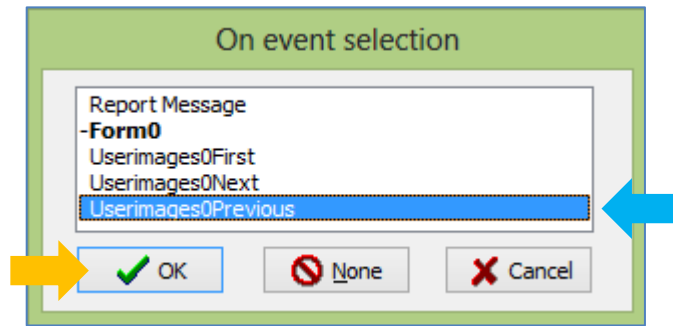


Linking Objects

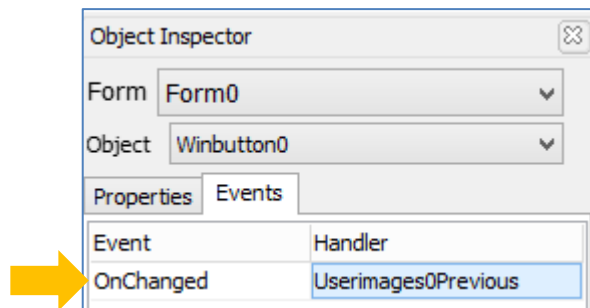
With Winbutton0 still selected, go to the object inspector and click on the ellipsis dots of the OnChanged event property. OnChanged is under the Events pane.



The On Event Selection window appears. Choose “Userimages0Previous” and click OK.



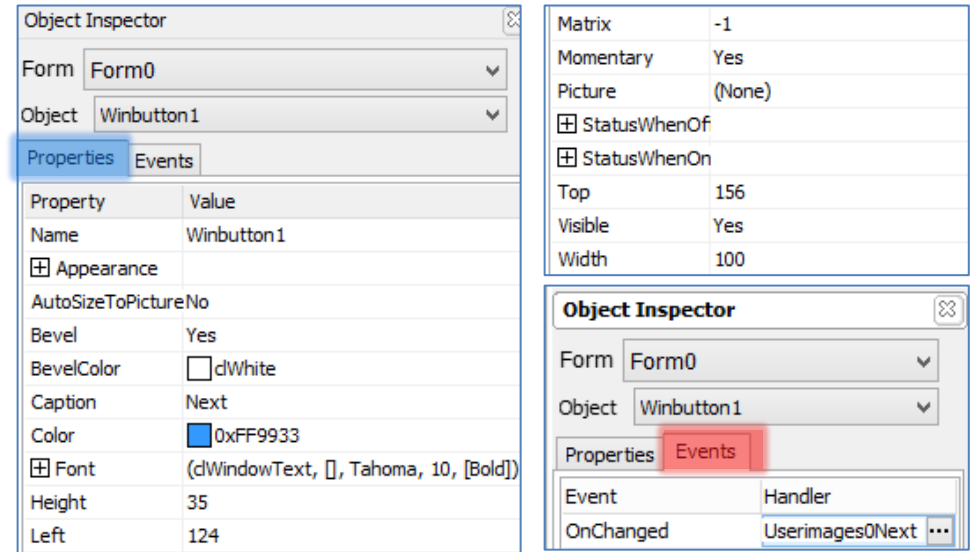
The OnChanged event property is updated.



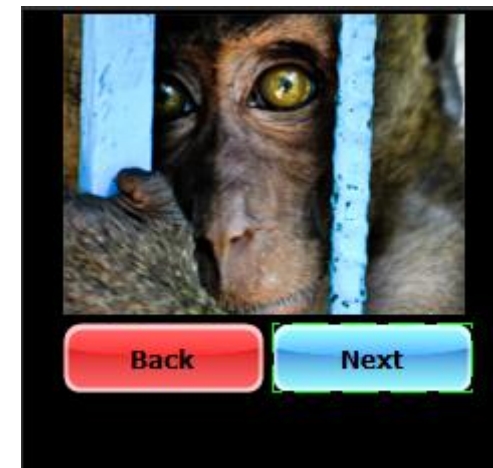
When the program runs, touching Winbutton0 will cause Userimages0 to display the previous frame or image.

Creating the “Next” Button

Add another win button to the project – Winbutton1. Here, Winbutton1, which has the following properties, is configured for displaying Userimages0’s next frame or image.

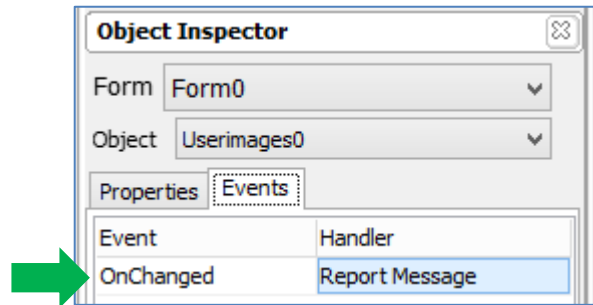


Don’t forget to configure Winbutton1’s OnChanged event property. The updated WYSIWYG screen is shown below.



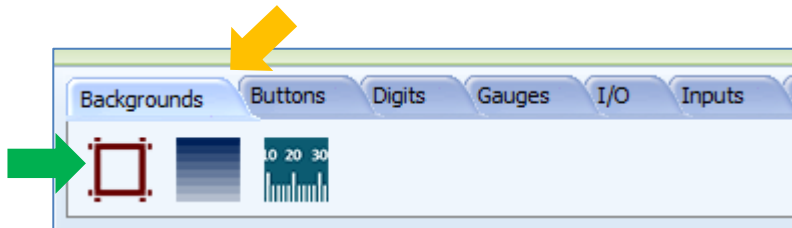
Configuring the User Images Object to Report a Message

The user images object has one single event, `onChanged`, which is useful for sending the value of the current frame. Go to the Events pane of `Userimages0`'s object inspector and configure the `OnChanged` event as shown below.

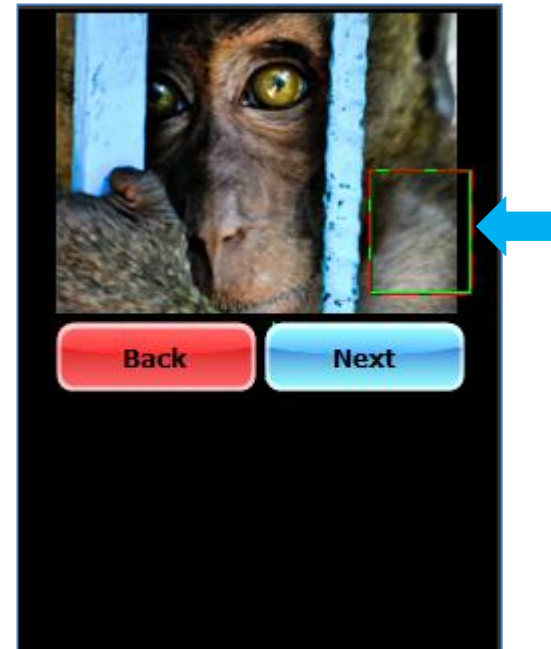


Adding a Border Background

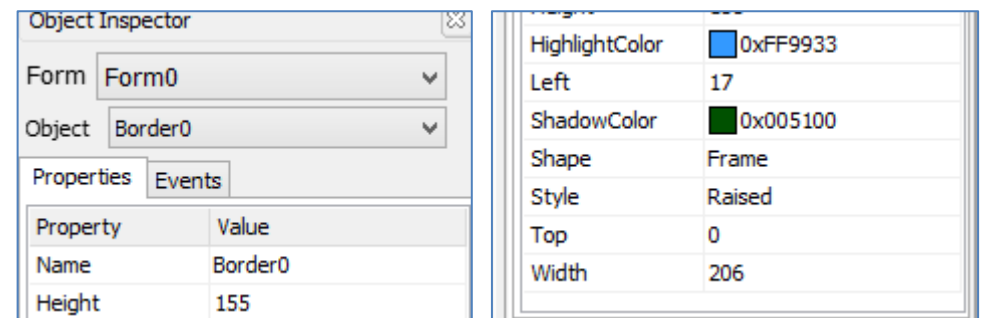
To add a border, go to the **Backgrounds** pane and click on the **border** icon.



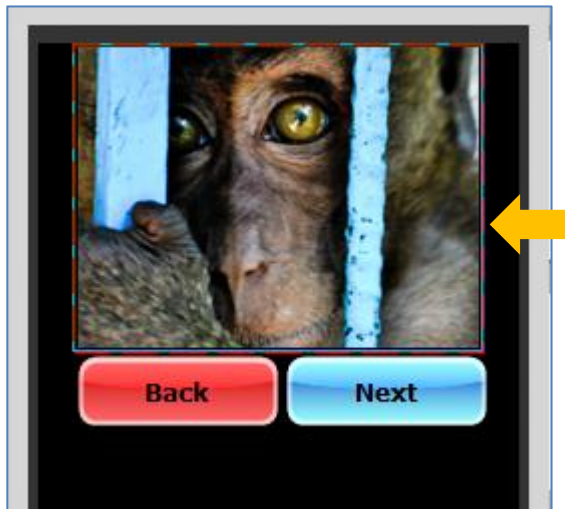
Click on the WYSIWYG screen to place the object.



The **Object Inspector** on the right part of the screen displays all the properties of the newly created border named **Border0**. The background border used in this project has the following properties.



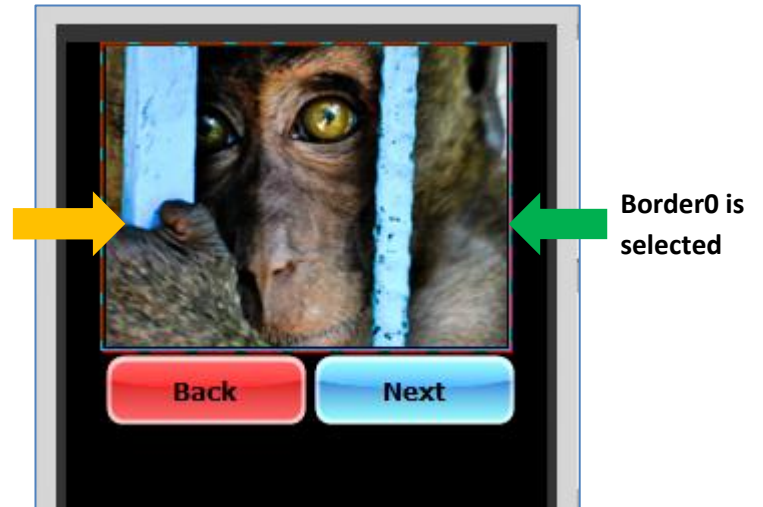
Shown below is the final appearance of the border background object.



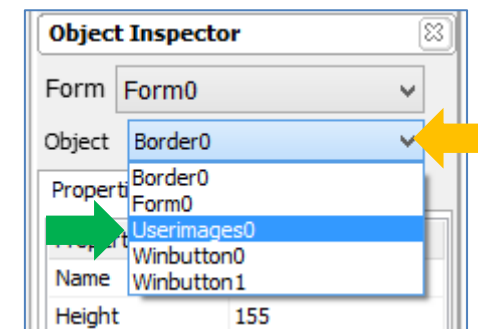
Selecting Objects

To inspect or edit the properties of an object, simply click on it in the WYSIWYG screen. However when one of two objects occupying a common area on the WYSIWYG screen is selected (a background object and a foreground object for example), it may be difficult to select the other object. An example of this case is the background border behind the user images object in this project. To illustrate:

Userimages0
is difficult to
select



To select a “hidden” object, simply go to the object inspector of the currently selected object, and click on the drop-down arrow of the object line.



Observe that the drop-down menu lists all the objects. Click on an object (Userimages0 for example) to select it.

Testing the Slideshow

The button-controlled slideshow is now complete. At this point the user has the option of building and loading the project to the display module. To test the slideshow, jump to the section “**Build and Upload the Project**”. The next section shows how to make a user images object behave like a video player.

Creating a Timer-driven Slideshow

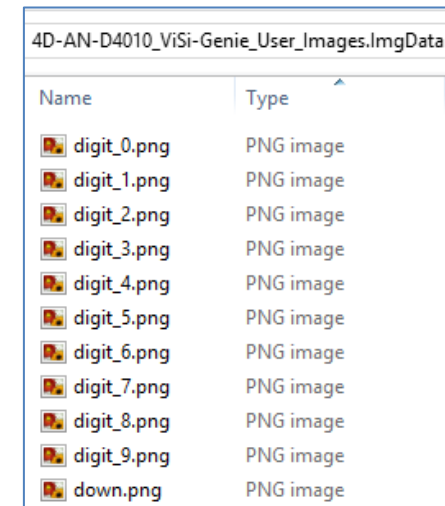
Besides being controlled by buttons, a user images object can also be controlled by a timer object. When the timer runs, the images are sequentially displayed. The timer object, in turn, can be started and stopped by buttons.

Add another User Images Object

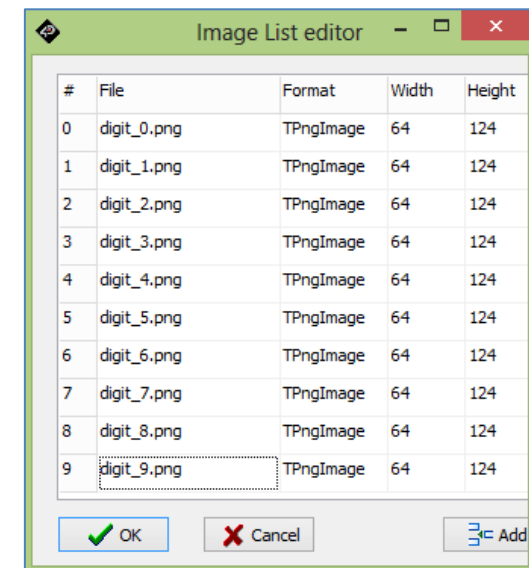
Another user images object (Userimages1) is added to the project. It is composed of ten images (digits 0 to 9).



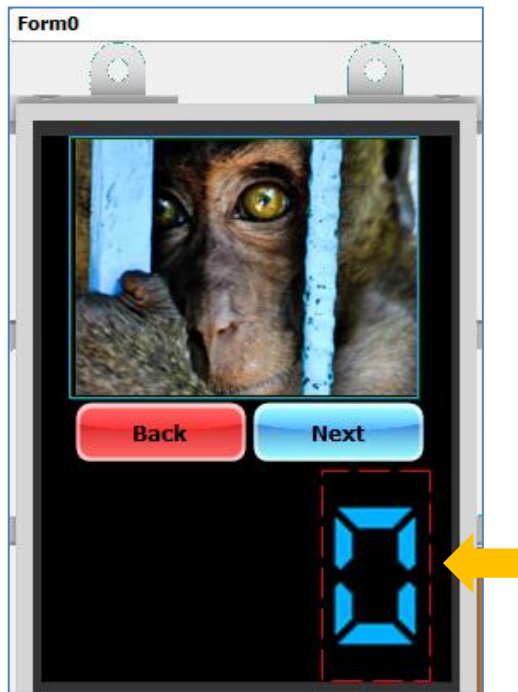
The image files for Userimage1 are in the “.ImgData” folder which is automatically generated when the demo project is compiled.



The Image List editor window shows the arrangement of the images.

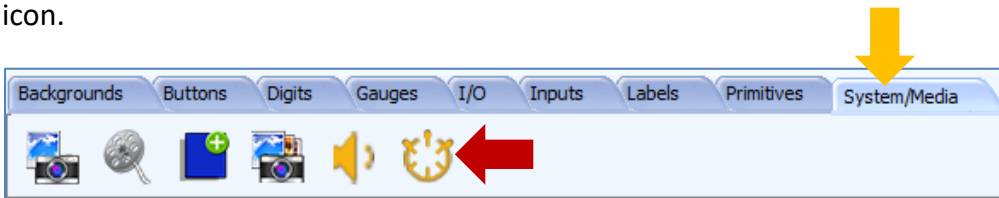


The WYSIWYG screen is updated accordingly.



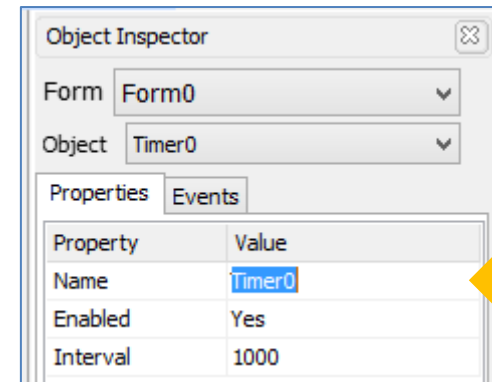
Adding a Timer Object

To add a timer object, go to the System/Media pane and click on the timer icon.

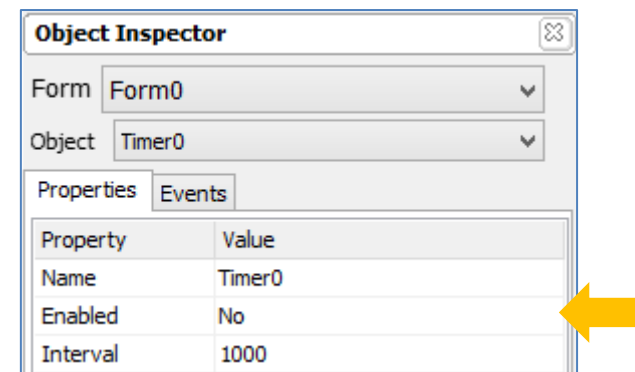


There is no need to click on the WYSIWYG screen to place the timer object. The timer object is automatically added to Form0 once it is selected on the System/Media pane. Note that timer objects (and the sounds object) always

reside on Form0. The object inspector shows the properties of the newly-created timer object named Timer0.



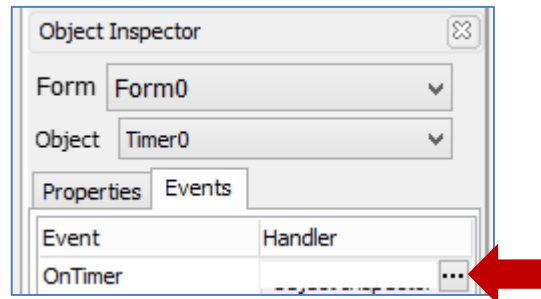
Note that the default value of the property “Interval” is 1000 (milliseconds). Set the property “Enabled” to “No”.



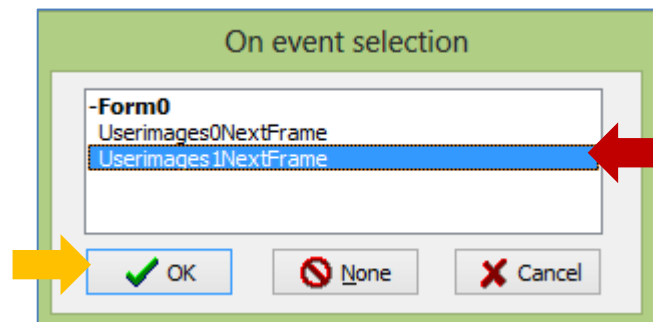
Setting the value of the property “Enabled” to “Yes” will cause the timer to start ticking once the program starts running on the display module. Setting the value to “No” disables this “auto-run” feature; the timer will wait for a triggering event instead.

Configuring a Timer to Drive a User Images Object

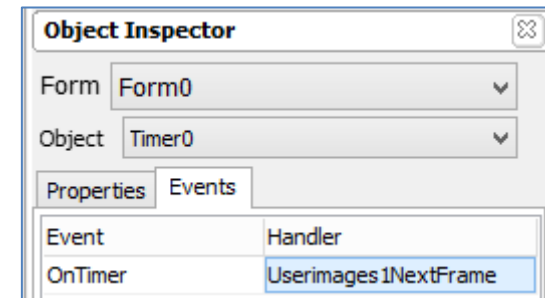
Click on the Events tab of Timer0 and click on the ellipsis dots of the OnTimer event.



The On event selection window appears. Select the option "Userimages1NextFrame" and click on the OK button.



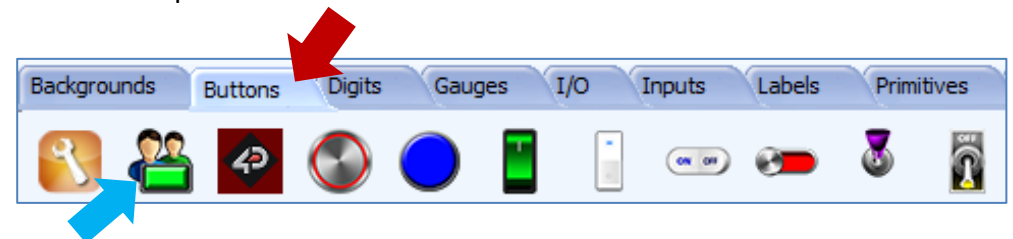
The value of the OnTimer property is updated.



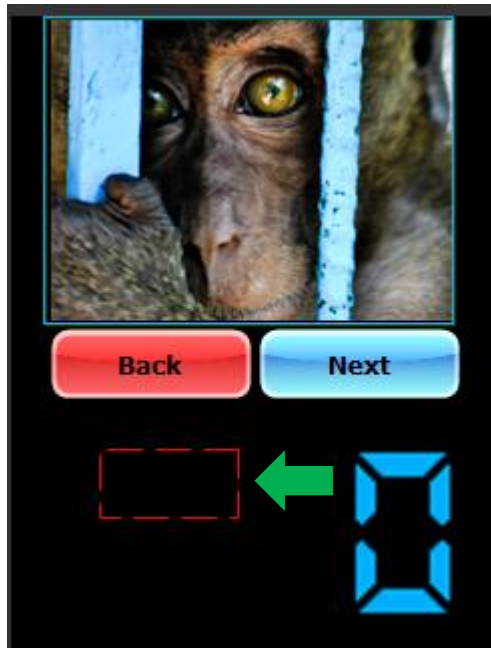
Now Timer0 will count down from 1000 milliseconds (or 1 second) to 0. Timer objects have a one-millisecond resolution. At 0 ms, it will trigger Userimages1 to display the next frame. Timer0 will repeat this process until all the frames of Userimages1 are displayed. Hence, it will take ten seconds for the timer-driven slideshow (Userimages1) to finish playing.

Add Momentary User Buttons to Control the Timer Object

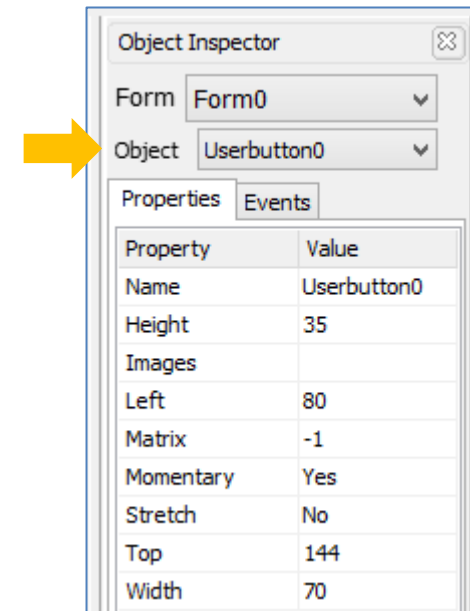
Two user buttons are added to be used as start and stop buttons for Timer0. Userbutton0 is the "start" button and Userbutton1 is the "stop" button. A key feature of user buttons is that the user has the option of providing the images to be used for each state. The user button widget icon is found under the Buttons pane. Click on it.



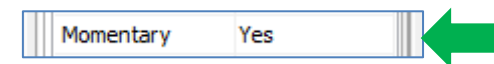
Click on the **WYSIWYG** (What-You-See-Is-What-You-Get) screen to put the object in place.



The empty object can be dragged to any desired location. The **Object Inspector** on the right part of the screen displays all the properties of the newly created user button object named **Userbutton0**.



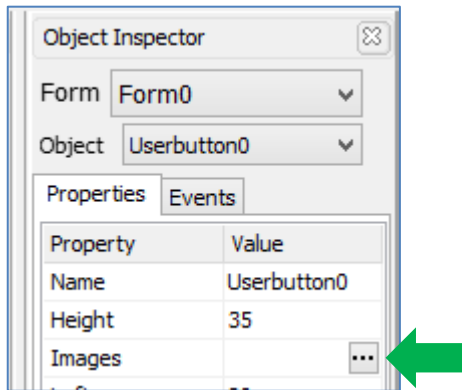
By default, a user button is a momentary button. Note that the property “Momentary” is set to “Yes”.



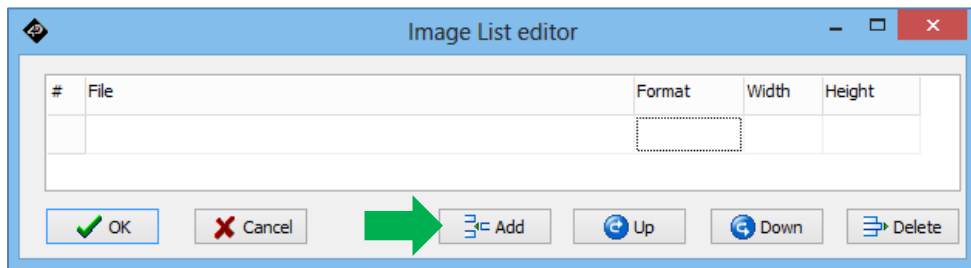
Take time to experiment with the different properties. To learn more about user buttons, refer to [ViSi-Genie User Button](#).

Defining the States of a Momentary User Button

A momentary button is enabled when pressed and disabled immediately upon release. It can be conveniently used for starting or stopping a timer. Here the momentary user button, Userbutton0, is used to start Timer0. To define the up and up pressed states, click on the ellipsis dots of the Images property line of the object inspector.

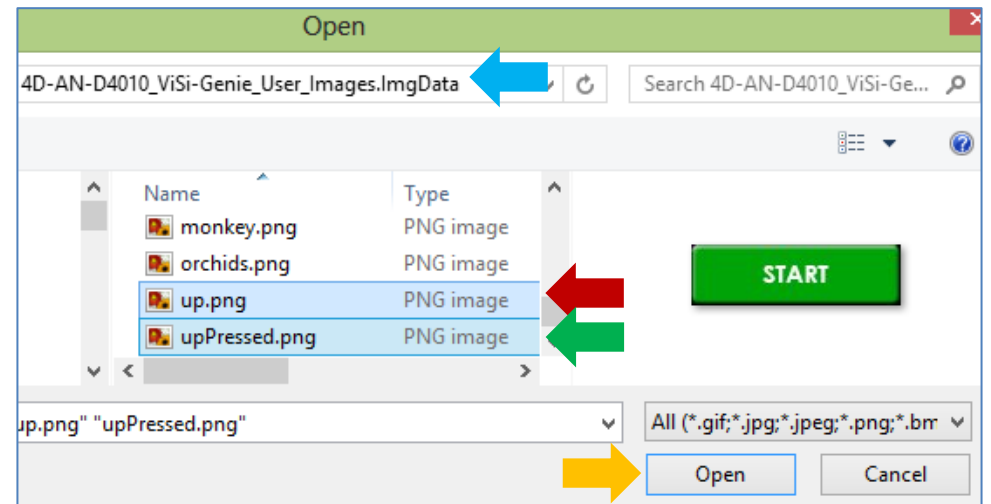


The Image List editor window appears. Click on the Add button to browse for the image files.

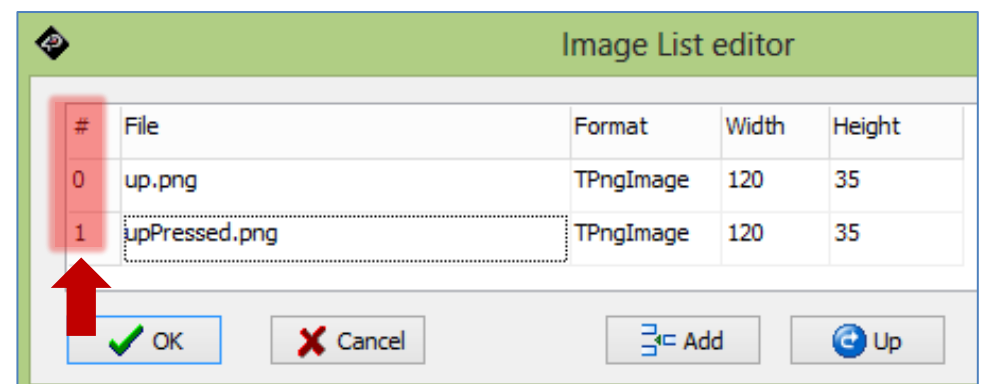


A standard open window appears and asks for image files. Multiple files can be selected. The figure below shows the two image files used for

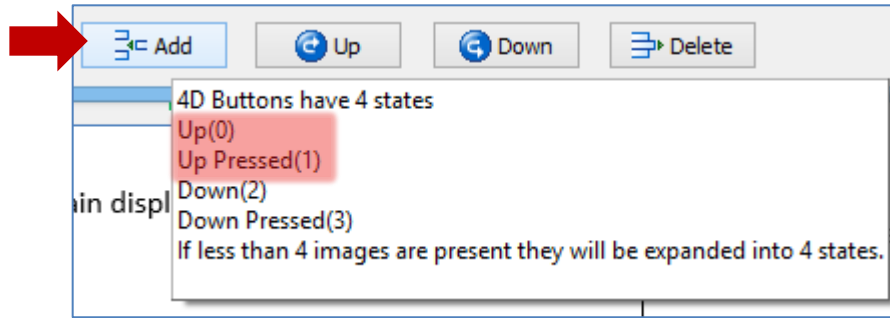
Userbutton0. All the image files used in this demo are saved in the **“.ImgData”** folder, which is automatically generated by Workshop when the project is compiled. After having selected the desired image files, click on the Open button.



The Image List editor window is again displayed.



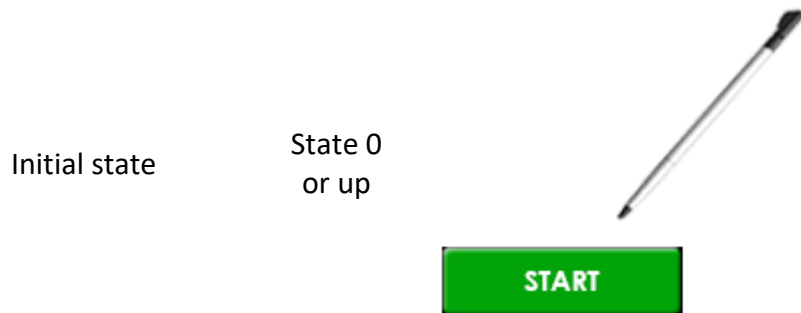
The left-most column lists the image numbers or the states. For additional information, hover over the Add button:



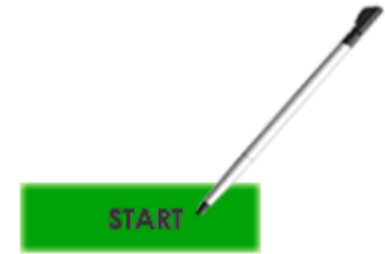
Note that Userbutton0 is a momentary button. Hence, it only has two states – ‘up’ and ‘up pressed’.



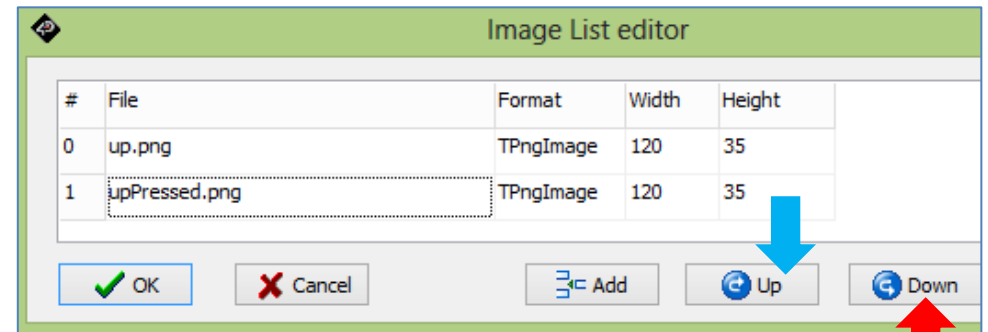
To illustrate:



Button is pressed State 1 or up pressed



To rearrange the order, select an image and click on the Up or Down button.



After having arranged the images to the desired order, click on the OK button. The WYSIWYG screen is updated with the initial state displayed.



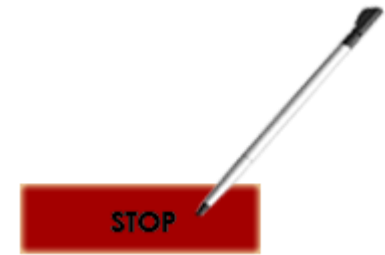
Add another user button, Userbutton1. This will be the stop button for Timer0. Below is the button in action.

Initial state

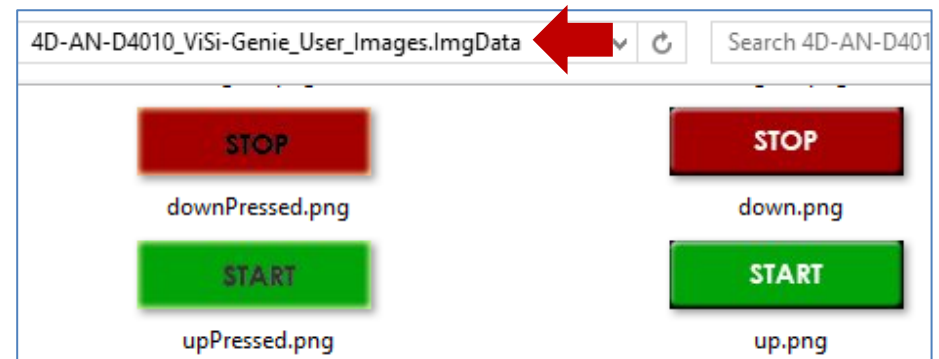
State 0
or up



Button is pressed
State 1 or
up pressed

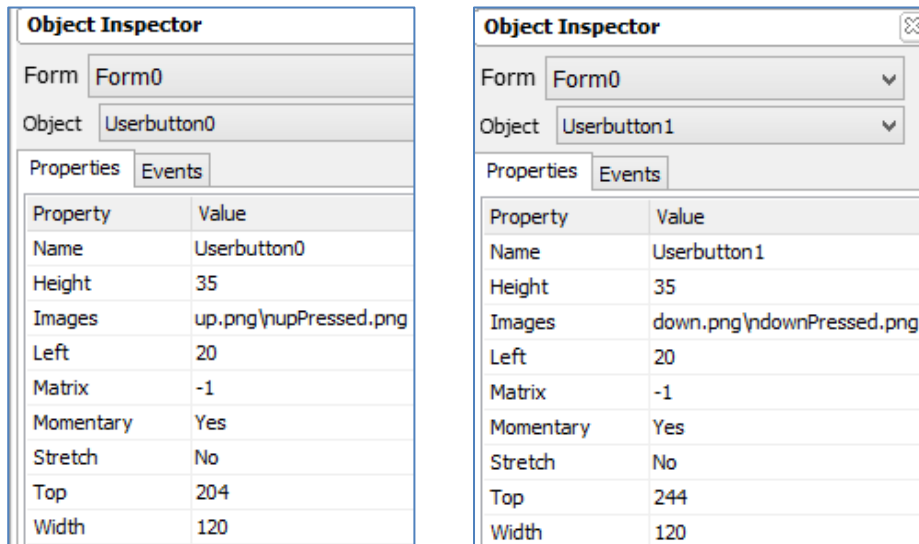


The image files for Userbutton0 and Userbutton1 are located in a common folder.



Shown below are the final properties of Userbutton0 and Userbutton1.

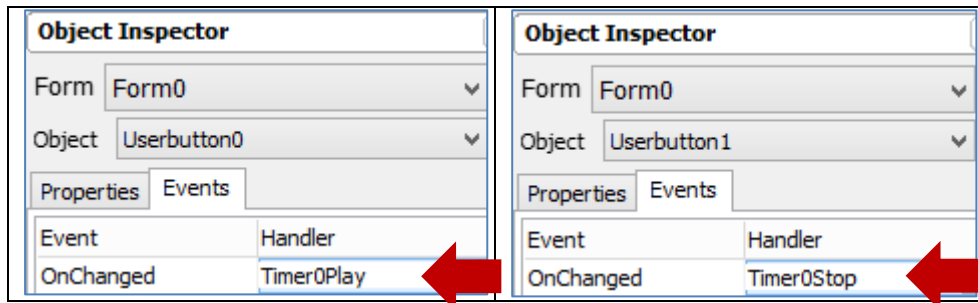




To know more about user buttons, refer to [ViSi-Genie User Button](#)

Link the User Buttons to Timer0

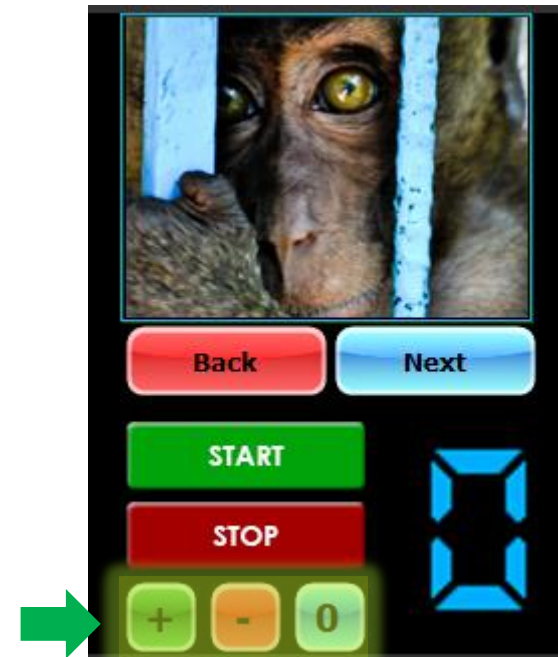
The OnChanged event property of Userbutton0 and Userbutton1 are configured to start and stop Timer0, respectively.



Add Three Win Buttons to Control UserImages1

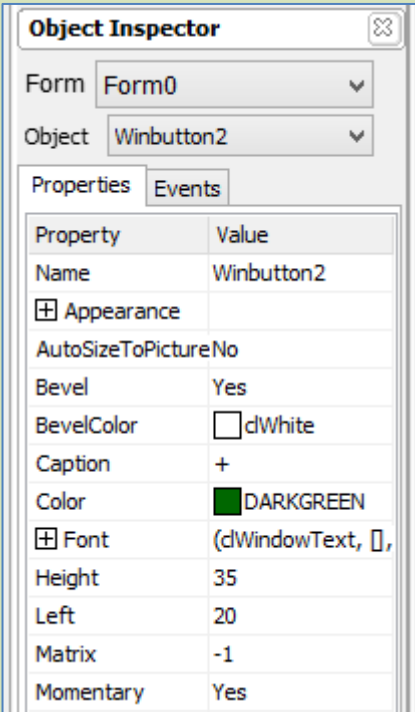
Additional control over UserImages1 is made possible with the use of three more win buttons. The user can make Userimages1 display the next,

previous, or first frame anytime. This is possible whether or not Timer0 is running.



The additional control win buttons have the following properties.

Winbutton2

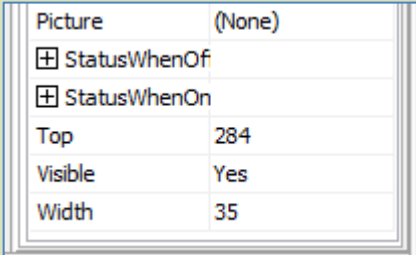


Object Inspector

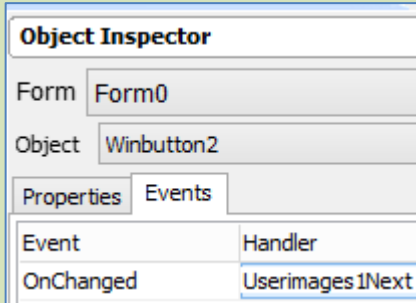
Form: Form0

Object: Winbutton2

Property	Value
Name	Winbutton2
Appearance	
AutoSizeToPictureNo	
Bevel	Yes
BevelColor	dWhite
Caption	+
Color	DARKGREEN
Font	(dWindowText, [], Tal)
Height	35
Left	20
Matrix	-1
Momentary	Yes



Picture	(None)
StatusWhenOf	
StatusWhenOn	
Top	284
Visible	Yes
Width	35




Object Inspector

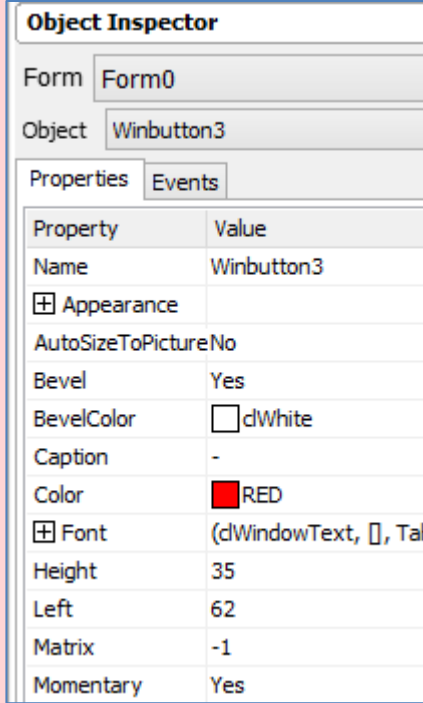
Form: Form0

Object: Winbutton2

Event	Handler
OnChanged	Userimages1Next



Winbutton3

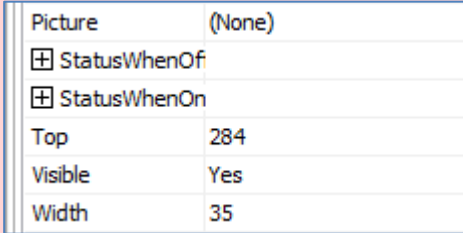


Object Inspector

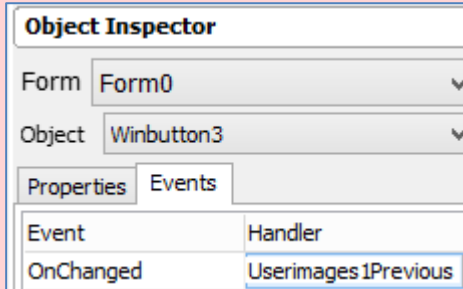
Form: Form0

Object: Winbutton3

Property	Value
Name	Winbutton3
Appearance	
AutoSizeToPictureNo	
Bevel	Yes
BevelColor	dWhite
Caption	-
Color	RED
Font	(dWindowText, [], Tal)
Height	35
Left	62
Matrix	-1
Momentary	Yes



Picture	(None)
StatusWhenOf	
StatusWhenOn	
Top	284
Visible	Yes
Width	35




Object Inspector

Form: Form0

Object: Winbutton3

Event	Handler
OnChanged	Userimages1Previous




Winbutton4

Object Inspector	
Form	Form0
Object	Winbutton4
Properties	
Property	Value
Name	Winbutton4
Appearance	
AutoSizeToPictureNo	
Bevel	Yes
BevelColor	<input type="checkbox"/> dWhite
Caption	0
Color	<input type="checkbox"/> 0xFF9933
Font	(dWindowText, [], Tah
Height	35
Left	104
Matrix	-1
Momentary	Yes

Picture	(None)
StatusWhenOf	
StatusWhenOn	
Top	284
Visible	Yes
Width	35

Object Inspector	
Form	Form0
Object	Winbutton4
Properties	
Events	
Event	Handler
OnChanged	Userimages1First



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

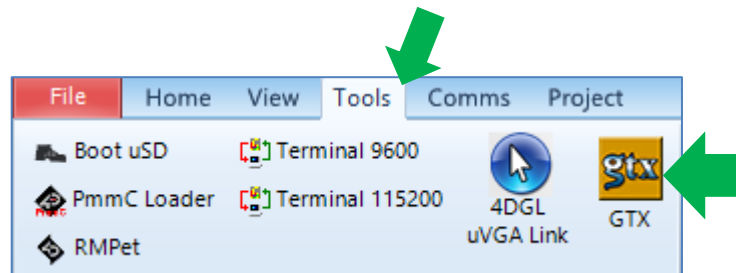
The display module is going to receive and send messages from and to an external host. This section explains to the user how to interpret these messages. An understanding of this section is necessary for users who intend to interface the display to a host. The [ViSi Genie Reference Manual](#) is recommended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

Use the GTX Tool to Analyse the Messages

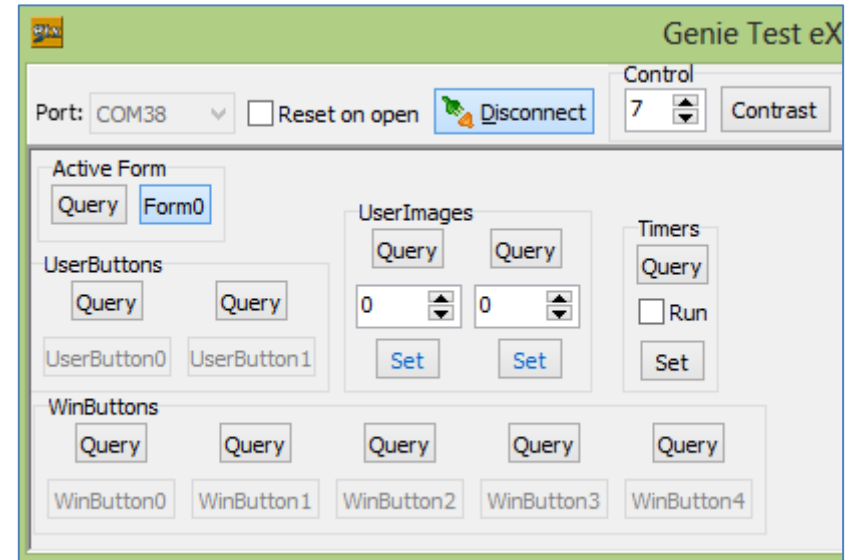
Using the GTX or **Genie Test eXecutor** tool is the first option to get the messages sent by the screen to the host. Here the PC will be the host. The GTX tool is a part of the Workshop 4 IDE. It allows the user to receive, observe, and send messages from and to the display module. It is an essential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

Launch the GTX Tool

Under the Tools menu click on the GTX tool button.



A new window appears showing all the objects of the project.



The User Images Object

Report Event

When the program starts, press and release Winbutton1 on the display module screen. Remember that Userimages0 was configured earlier to report a message when its status has changed.



Upon releasing Winbutton1, the second frame of Userimages0 is displayed. Also, a message is sent from the display module to the PC. The message is displayed on the white area on the right part of the GTX Tool window.

```
Winbutton Change 11:22:12.177 [07 06 01 00 01 01]
```

The actual message bytes are those inside the brackets. These values are in hexadecimal. The figure preceding the actual message is the computer time at which the message is sent. A label is also included to tell the observer what the message represents.



The message received is formatted according to the following pattern:

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
07	06	01	00	01	01
REPORT_EVENT	Win button	Second	0x0001		

The message is from Winbutton1, and it contains the hexadecimal value “0x0001”. Normally the value (MSB and LSB) contained in a message coming from a button reflects the current status of that button. For example, a value of 0x0001 means that the button is enabled; a value of 0x0000 means that the button is disabled. Moreover, the message can be any of the following types: a REPORT_EVENT (the button was configured to report a message when its status has changed), a REPORT_OBJ (the button is responding to a request for status from the host), or a WRITE_OBJ (the host is writing to the button to change its state). A special case arises however when a button is used to control a user images (or a video) object and that user images (or video) object is configured to report a message when its status has changed. In this situation, the value of a REPORT_EVENT message will always reflect the most current frame of the user images (or video) object to which the button is linked. To illustrate, press and release Winbutton1 three times (or

until the fourth frame of Userimages0 is displayed). The received messages from the display module are shown below.

```
Winbutton Change 12:04:16.442 [07 06 01 00 01 01]
Winbutton Change 12:04:17.409 [07 06 01 00 02 02]
Winbutton Change 12:04:18.242 [07 06 01 00 03 03]
```

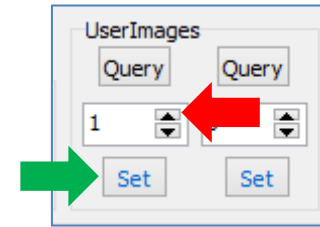
Note that the value of the message increases from 0x0001 to 0x0003, which means that the frames of Userimages0 are sequentially displayed from frame 1 to frame 3. Pressing and releasing Winbutton0 (the “Back” button) will send the following messages:

```
Winbutton Change 12:05:56.553 [07 06 00 00 02 03]
Winbutton Change 12:05:57.367 [07 06 00 00 01 00]
Winbutton Change 12:05:58.178 [07 06 00 00 00 01]
```

The checksum is a means for the host to verify if the message received is correct. This byte is calculated by XOR’ing all bytes in the message from (and including) the CMD or command byte to the last parameter byte. Then, the result is appended to the end to yield the checksum byte. If the message is correct, XOR’ing all the bytes (including the checksum byte) will give a result of zero. Checking the integrity of a message using the checksum byte shall be handled by the host.

Control a User Images Object using the GTX Tool

In the GTX tool window, set the desired frame number and click on the Set button as shown below.



Note that frame 1 (the second frame) of Userimages0 is shown on the display module screen. Also, the white area on the right displays

- in **green** the messages sent to the display module
- and in **red** the messages received from the display module

```
Set UserImages Value 13:36:29.999 [01 1B 00 00 01 1B]
ACK 13:36:30.077 [06]
```

The message sent is formatted according to the following pattern:

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
01	1B	00	00	01	1B
WRITE_OBJ	User images	First	0x0001		

The message stands for “Write to the first user images object on the display module the value **0x0001**”.

ACK = 0x06 as shown below

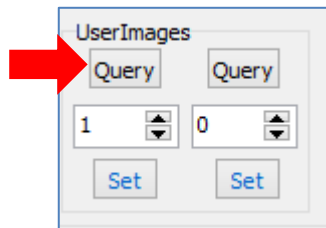
```
ACK 08:28:01.868 [06]
```

is an acknowledgment from the display module which means that it has understood the message.

Polling a User Images Object

If a user images object was not configured to report a message when its status has changed, it is still possible to know which frame is currently displayed by polling.

To do this, click on the Query button.



Messages are sent to and received from the display module.

```
Request UserImages Value 13:41:07.472 [00 1B 00 1B]
UserImages Value 13:41:07.503 [05 1B 00 00 01 1F]
```

The messages are formatted according to the following pattern:

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum
00	1B	00	-	-	1B
READ_OBJ	User images	First	N/A		
05	1B	00	00	01	1F
REPORT_OBJ	User images	First	0x0001		

The host sends a READ_OBJ command specifically asking for the frame value of the first user images object. The display module then responds with the current frame value of that user images object. Communication between a 4D display module programmed with a ViSi-Genie application and an external host controller must follow the ViSi-Genie Communications Protocol, which is defined in the [ViSi Genie Reference Manual](#).

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.