



# ViSi Displaying Images from The uSD Card – WYSIWYG FAT16

DOCUMENT DATE: **21<sup>st</sup> MAY 2019**  
DOCUMENT REVISION: **1.1**



## Description

This application note shows how to add image and video objects in ViSi. Before getting started, the following are required:

- Any of the following 4D Picaso and gen4 Picaso display modules:

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

and other superseded modules which support the ViSi Genie environment

- The target module can also be a Diablo16 display

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

Visit [www.4dsystems.com.au/products](http://www.4dsystems.com.au/products) to see the latest display module products that use the Diablo16 processor. The display module used in this application note is the uLCD-32PTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.

- [4D Programming Cable](#) / [uUSB-PA5/uUSB-PA5-II](#) for non-gen4 displays(uLCD-xxx)
- [4D Programming Cable](#) & [gen4-PA](#) / [gen4-IB](#) / [4D-UPA](#)

for gen4 displays (gen4-uLCD-xxx)

- [micro-SD \(μSD\)](#) memory card
- [Workshop 4 IDE](#) (installed according to the installation document)
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

## Content

<b>Description</b> .....	<b>2</b>
<b>Content</b> .....	<b>3</b>
<b>Application Overview</b> .....	<b>3</b>
<b>Setup Procedure</b> .....	<b>4</b>
<b>Create a New Project</b> .....	<b>4</b>
<b>Design the Project</b> .....	<b>4</b>
<i>Uncomment the uSD Card Initialization Routine</i> .....	<b>4</b>
<i>Image Object</i> .....	<b>5</b>
Add an Image Object to the WYSIWYG Screen .....	<b>5</b>
Insert the Image Object Code .....	<b>6</b>
<i>Video Object</i> .....	<b>7</b>
Add a Video Object to the WYSIWYG Screen .....	<b>7</b>
Video Frames .....	<b>7</b>
Insert the Video Object Code .....	<b>8</b>
An Example .....	<b>10</b>
<b>Run the Program</b> .....	<b>11</b>
<i>Workshop Sample Programs</i> .....	<b>11</b>
<b>Proprietary Information</b> .....	<b>13</b>
<b>Disclaimer of Warranties &amp; Limitation of Liability</b> .....	<b>13</b>

## Application Overview

ViSi allows the user to place a widget (an image object for example) on the WYSIWYG (What-You-See-is-What-You-Get) screen and paste the corresponding 4DGL code to the code area. Workshop 4 then automatically generates the graphics files in the background and copies them to the uSD card. The 4DGL code is compiled and downloaded to the display module. When the program runs, it accesses the graphics files on the uSD card and displays the desired object on the screen.

This application note explains how to create image and video objects in the WYSIWYG screen, how to insert the generated object codes into the main program, and how to display the objects using simple 4DGL commands. For video objects, this tutorial shows how video frames are displayed either one at a time or continuously.

## Setup Procedure

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section “**Setup Procedure**” of the application note

[ViSi Getting Started - First Project for Picaso and Diablo16](#)

## Create a New Project

For instructions on how to create a new **ViSi** project, please refer to the section “**Create a New Project**” of the application note

[ViSi Getting Started - First Project for Picaso and Diablo16](#)

## Design the Project

### Uncomment the uSD Card Initialization Routine

Remove the block comment symbols as shown below.

```
11 func main()
12 // var hstrings ; // Handle to access uSD strings,
13 // var hFontx ; // Handle to access uSD fonts, un
14 // Uncomment the following if uSD images, fonts or
15 /*
16     putstr("Mounting...\n");
17     if (!(disk:=file_Mount()))
18         while(!(disk:=file_Mount()))
19             putstr("Drive not mounted...");
20             pause(200);
21             gfx_Cls();
22             pause(200);
23         wend
24     endif
25     gfx_TransparentColour(0x0020);
26     gfx_Transparency(ON);
27
28 // hFontn := file_LoadImageControl("NoName4.dan", "
29 // hstrings := file_Open("NoName4.txf", 'r') ; // C
30 hndl := file_LoadImageControl("NoName4.dat", "No
31 */
```

The code screen will be updated accordingly, showing the block as an actual part of the code for compilation.

```

11 func main()
12 // var hstrings ; // Handle to access uSD strings,
13 // var hFontx ; // Handle to access uSD fonts, un
14 // Uncomment the following if uSD images, fonts or
15
16     putstr("Mounting...\n");
17     if (!(disk:=file_Mount()))
18         while(!(disk:=file_Mount()))
19             putstr("Drive not mounted...");
20             pause(200);
21             gfx_Cls();
22             pause(200);
23     wend
24 endif
25 gfx_TransparentColour(0x0020);
26 gfx_Transparency(ON);
27
28 // hFontn := file_LoadImageControl("NoName4.dan", "
29 // hstrings := file_Open("NoName4.txf", 'r') ; // O
30 hndl := file_LoadImageControl("NoName4.dat", "No
31

```

Leave lines 28 and 29 as they are, since they are not needed in this application.

The function `file_LoadImageControl(fname1,fname2,mode)` in line 30 creates an image control list. It requires two files – `fname1` and `fname2`, the `.dat` file and `.gci` file, respectively. These files are created by Workshop. The GCI file contains all the graphics for the images and/or videos created by Workshop. The DAT file contains one line for each image or video, that

names the object and gives its starting offset within the GCI and its initial X/Y position. The function returns a handle (pointer to the memory allocation) to the image control list that has been created. This handle will be used to access and display objects, as will be shown later.

```

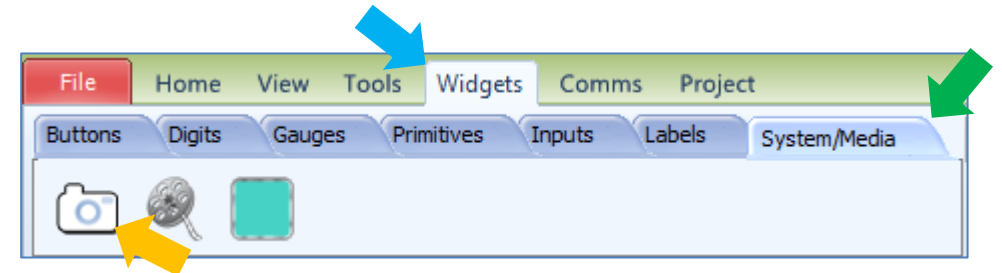
29 // hstrings := file_Open("NoName1.txf", 'r') ; // Open handle to access t
30 hndl := file_LoadImageControl("NoName1.dat", "NoName1.gci", 1);
31

```

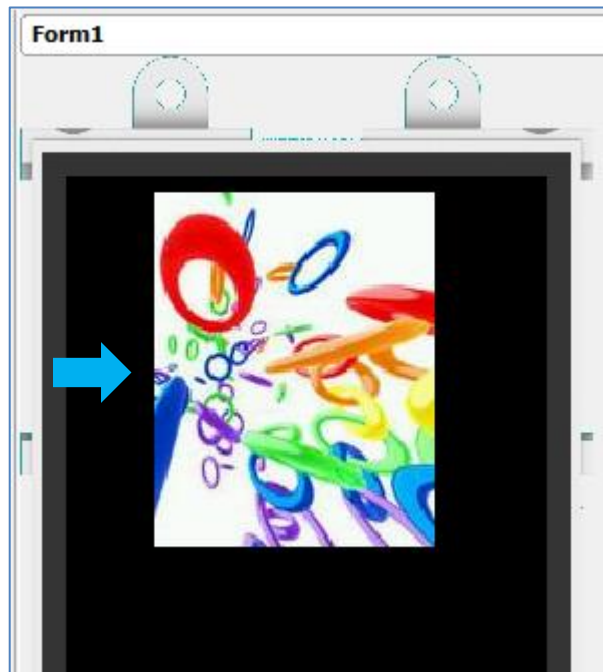
## Image Object

### Add an Image Object to the WYSIWYG Screen

Go to the Widgets menu, select the System/Media pane, and click on the Image icon.



Once the Image icon is selected, click on the WYSIWYG screen to place it. A standard Open window appears. Browse for the desired file and click Open. The WYSIWYG screen is updated.



The user can drag the object to any desired location in the WYSIWYG screen. It is also possible to resize the image by dragging the borders. Further editing of other properties can be done using the Object Inspector. The user may want to limit the image size (length and width) so as to give room for a video object later on in this application note.

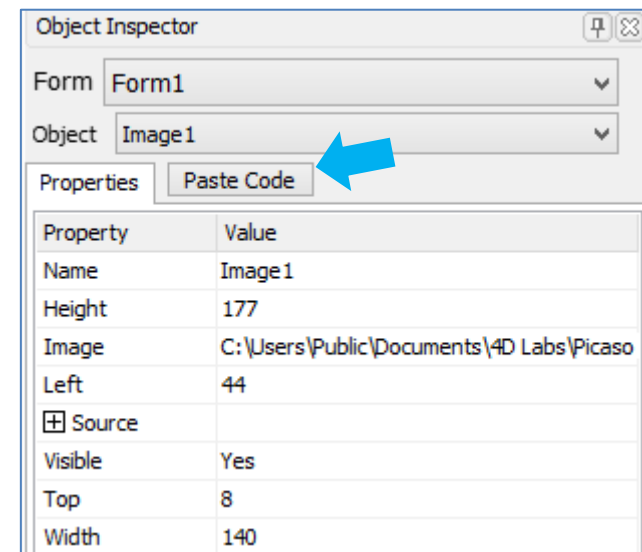
### Insert the Image Object Code

Go to the code area and place the cursor just after the handle assignment statement (line 32 in this example).

```

28 // hFontn := file_LoadImageControl ("NoName1.dar
29 // hstrings := file_Open ("NoName1.txf", 'r') ;
30 hndl := file_LoadImageControl ("NoName1.dat",
31
32 |
33
34     repeat
35     forever
36 endfunc
  
```

Having selected the image object, go to the Object Inspector and click on the Paste Code button.



The program code will be updated accordingly.

```

30  hndl := file_LoadImageControl("NoName1.dat", "NoName1.gci", 1);
31
32
33  // Image1 1.0 generated 6/8/2013 12:29:56 PM
34  img_Show(hndl,iImage1) ;
35
36
37  repeat
38  forever
39  endfunc
40

```

A new line for the image object is generated, along with a comment. The command

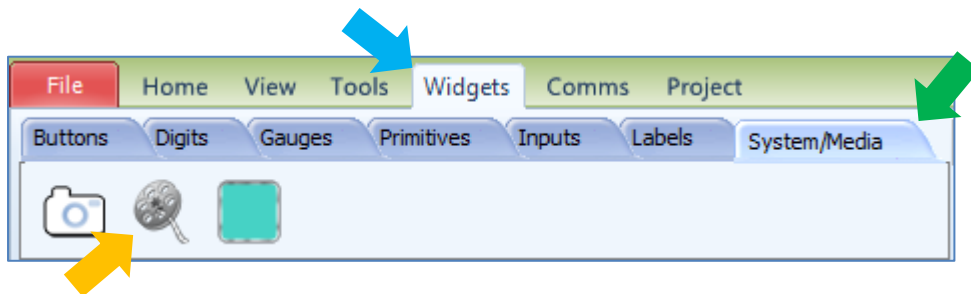
```
img_Show(hndl,iImage1)
```

simply displays the image that has been created. Note that **iimage1** is an image index for the object created.

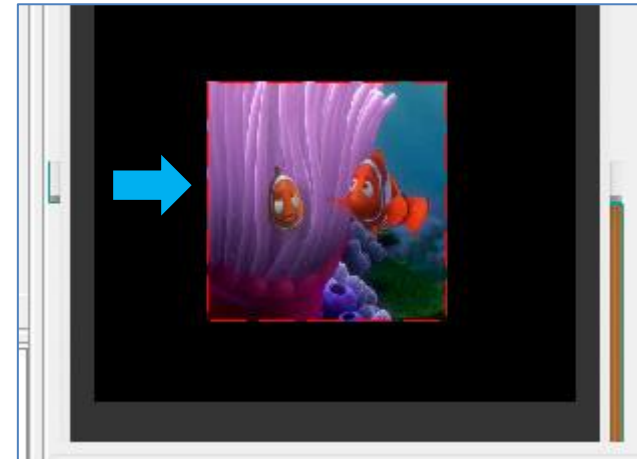
## Video Object

### Add a Video Object to the WYSIWYG Screen

Go to the Widgets menu, select the System/Media pane, and click on the Video icon.




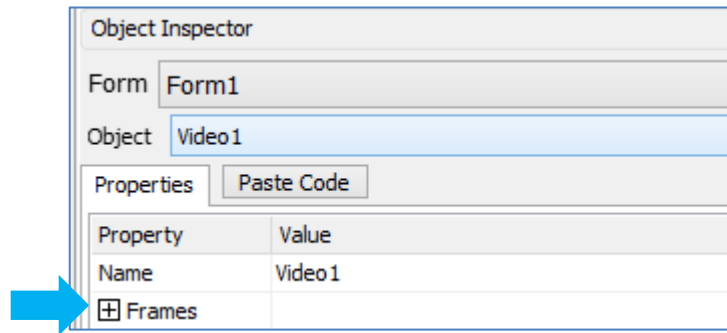
Once the Video icon is selected, click on the WYSIWYG screen to place it. A standard Open window appears and displays the files that can be opened. Browse for the desired file and click Open. The WYSIWYG screen is updated.



The user can drag the object to any desired location in the WYSIWYG screen. It is also possible to resize the object by dragging the borders. Further editing of other properties can be done using the Object Inspector.

### Video Frames

It is possible for the user to know the number frames inside a video. In the Object Inspector click on the symbol  beside Frames.




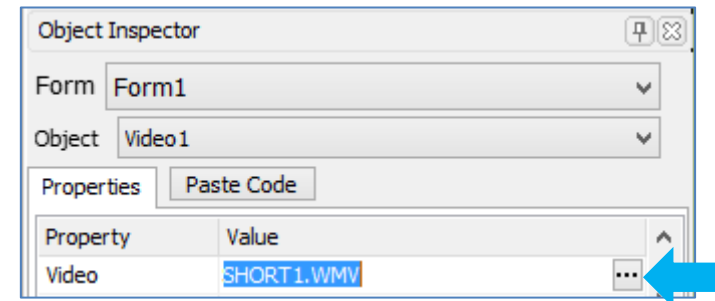
The properties under Frames will be displayed.

Property	Value
[-] Frames	
First	0
Last	604
FrameDelay	33

The video added in this application note has the frame properties shown above. Total number of frames is 605 – from 0 (First) to 604 (Last). The user has the option of choosing what frames to include in the GCI file by changing the First and Last frame properties. For example,

Property	Value
[-] Frames	
First	24
Last	567
FrameDelay	33

The user can view the actual frames by clicking on the Video property line and clicking on the symbol .



The Image + Video Converter window now appears. Refer to [ViSi-Genie Play Video](#) for detailed information on how to use the Image + Converter window. FrameDelay is the delay between each frame. The unit is in milliseconds. The reciprocal of FrameDelay is the frame rate of the video. For example, the video used in this application note has a frame rate of  $1/.033 = 30$  frames per second.

### Insert the Video Object Code

Go to the code area and place the cursor after the handle assignment statement (line 37 after the `img_Show()` command in this example).

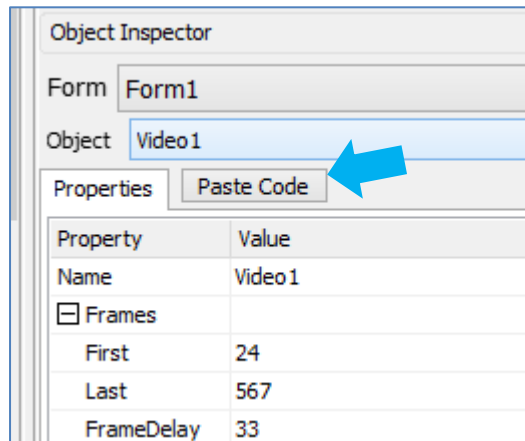
```

30 // hstrings := file_Open("VIDIMA~1.txf", 'r') ; // Open handle to ac
31 hndl := file_LoadImageControl("VIDIMA~1.dat", "VIDIMA~1.gci", 1);
32
33
34 // Image1 1.0 generated 6/8/2013 12:29:56 PM
35 img_Show(hndl,iImage1) ;
36
37 |
38 repeat
39 forever
40 endfunc

```



Having selected the video object, go to the Object Inspector and click on the Paste Code button.



The program code will be updated accordingly.

```

34 // Image1 1.0 generated 6/8/2013 12:29:56 PM
35 img_Show(hndl,iImage1) ;
36
37
38 // Video1 1.0 generated 6/8/2013 2:41:11 PM
39 img_SetWord(hndl, iVideo1, IMAGE_INDEX, frame) ;
40 img_Show(hndl,iVideo1) ;
41
42 repeat
43 forever
44 endfunc
45

```

A new block for the video object is generated, along with a comment. The command

```
img_SetWord(hndl, iVideo1, IMAGE_INDEX, frame)
```

is used to set which frame of the movie is to be displayed. The user can replace the fourth argument, **frame**, with a frame value. For instance, set the program to display frame 0 of the object Video1 by writing the statement

```
img_SetWord(hndl, iVideo1, IMAGE_INDEX, 0)
```

After having set the frame to be shown, display the frame with the command

```
img_Show(hndl,iVideo1)
```

To display three frames in succession, the user can use the code below.

```

img_SetWord(hndl, iVideo1, IMAGE_INDEX, 0) ;
img_Show(hndl,iVideo1) ;
pause(1000); //add a one second delay

img_SetWord(hndl, iVideo1, IMAGE_INDEX, 1) ;
img_Show(hndl,iVideo1) ;
pause(2000); //add a two-second delay

img_SetWord(hndl, iVideo1, IMAGE_INDEX, 2) ;
img_Show(hndl,iVideo1) ;
pause(3000); //add a three-second delay

```

Note that for the fourth argument, **frame**, the range of possible values depends on how many frames the video object has - 0 to 604 for the video

used in this application note. If the user is doing pure codes in the Designer environment, is also possible to get the number of frames by using the function

```
img_GetWord(hndl, iVideo1, IMAGE_FRAMES) ;
```

### An Example

To play an entire video, the user can use a loop with an incrementing counter. To illustrate:

```
for(frame := 0; frame <= 604; frame ++)  
    img_SetWord(hndl, iVideo1, IMAGE_INDEX, frame) ;  
    img_Show(hndl,iVideo1) ;  
    pause(33);  
next
```

Or

```
n := img_GetWord(hndl, iVideo1, IMAGE_FRAMES) ;  
for(frame := 0; frame < n; frame ++)  
    img_SetWord(hndl, iVideo1, IMAGE_INDEX, frame) ;  
    img_Show(hndl,iVideo1) ;  
    pause(33);  
next
```

Note that the delay used is 33 milliseconds, which is equal to the FrameDelay property of the video file. This enables the program to play the video at its original frame rate.

This application note comes with a ViSi file, **VIDIMAGEtutorial**, which contains a code for playing an entire video. It also contains the codes shown in the previous sections of this tutorial.

## Run the Program

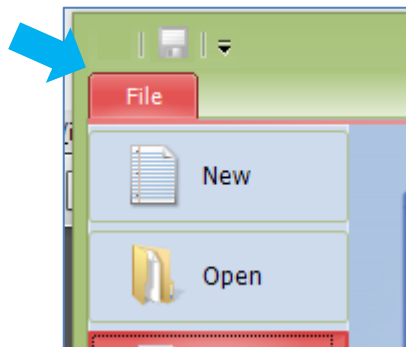
For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination (this option is not available for Goldelox displays), and how to compile and download a program, please refer to the section “**Run the Program**” of the application note

### [ViSi Getting Started - First Project for Picaso and Diablo16](#)

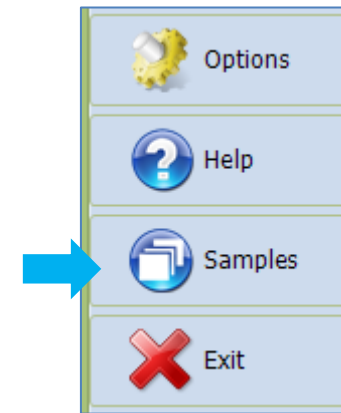
The uLCD-32PTU and uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

### Workshop Sample Programs

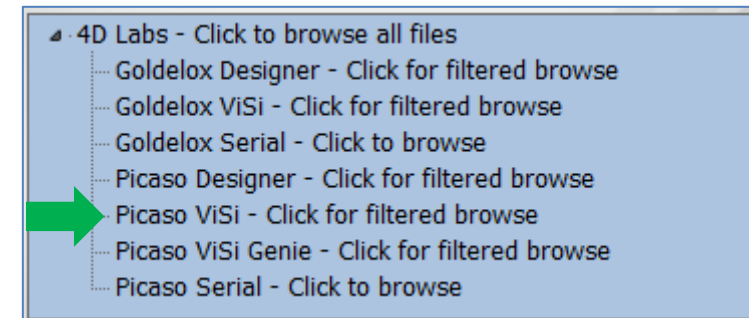
Workshop comes with many sample programs, one of which is entitled **VIDIMAGE**. This program is almost identical to VIDIMAGEtutorial. To open it, click on the File menu tab.



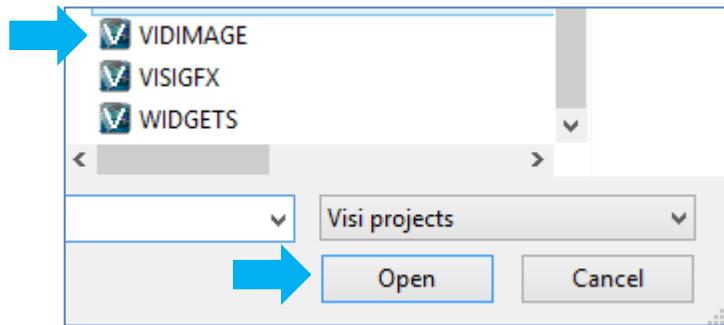
Near the bottom of the drop down menu, you can find the Samples button, click on it.



The samples window now appears. Select Picaso ViSi.



Select the file VIDIMAGE then click on Open.



The program now opens.

## 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.