APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

General Downloading an Application Program to RAM or Flash Memory

DOCUMENT DATE: **1st MAY 2020**
DOCUMENT REVISION: **1.3**

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:

- Any of the following 4D Picaso touch display modules:

gen4-uLCD-24PT	gen4-uLCD-28PT	gen4-uLCD-32PT
uLCD-24PTU	uLCD-28PTU	

other superseded modules which support the ViSi-Genie environment

Visit www.4dsystems.com.au/products to see the latest Picaso display products.

- 4D Programming Cable or µUSB-PA5/uUSB-PA5-II
- 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 discussed in these recommended application notes.

Content

Application Overview

When downloading (or uploading) a ViSi or Designer program to the processor of a display module, the user has a choice between two destinations - the RAM (Random Access Memory) or the flash memory section. Furthermore, when a program is downloaded (or uploaded) to the flash memory, the user can choose whether to make it run from where it resides (run from flash) or from RAM (run from RAM). This application note explains the difference between the two destinations and why the user would choose one over the other.

For the ViSi-Genie environment, programs are always downloaded to the flash memory. The user has the option of making the program run from the flash memory (run from flash) or from RAM (run from RAM). For more information, refer to the application note ViSi-Genie Program Destination.

For the Serial environment, the SPE application is automatically downloaded to the flash memory, and the RAM option is not available. Also, the SPE application residing in the flash memory always runs from RAM.

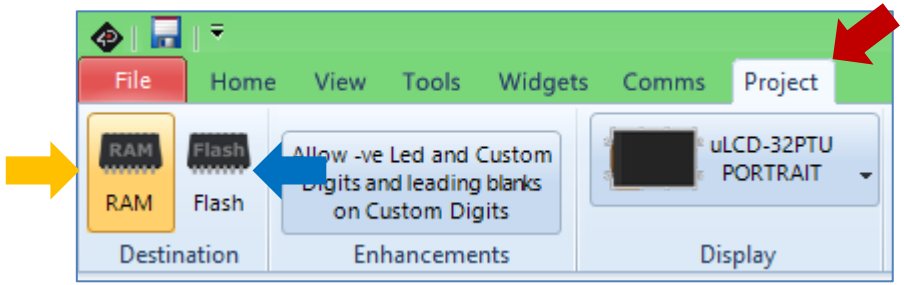
Program Destination

Allocation Size of Flash and RAM

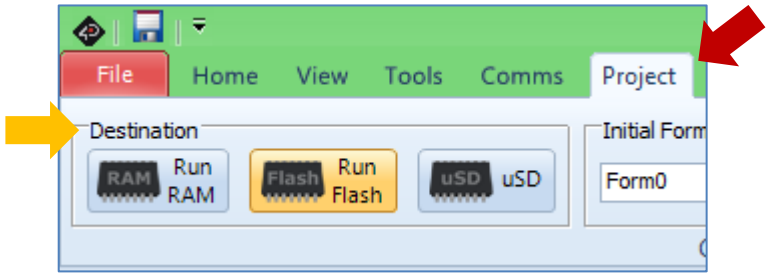
The Goldelox processor has 10KB of flash memory and 510 bytes of RAM. The Picaso processor has 14KB of flash memory and 14KB of RAM. The Diablo16 processor has 32KB of RAM and six banks of flash memory – Bank 0 to Bank 5. Each bank of flash memory is 32 KB. By default, programs are downloaded to Bank 0.

The RAM and Flash Destination Buttons

The program destination buttons are found under the Project menu or tab.



To make Workshop download the program to RAM, choose “RAM”. To make Workshop download the program to the flash memory, choose “Flash”. The RAM and Flash buttons are available in Designer and ViSi only. In ViSi-Genie, programs are automatically downloaded to the flash memory. There are no buttons that allow the user to choose between RAM or flash memory as the program destination in the ViSi-Genie environment. The same is the case with the Serial environment. The SPE application is automatically downloaded to the flash memory, and the RAM option is not available.
In ViSi-Genie, the Project menu or tab also has a Destination group.



Note however that the first two buttons are labelled as “Run RAM” and “Run Flash”. These have a different purpose, as explained in detail in the application note ViSi-Genie Program Destination.

Choosing between Flash and RAM as the Destination

While still in the process of developing a Visi or Designer application, it is best to download the program straight to RAM. When the program is finalized, it can then be downloaded to the flash memory. It is best to avoid frequent use of the flash memory as the program destination because it has a life cycle time, which means that it can only be written to a finite number of times. This figure extends beyond tens of thousands of times though, and it should not be considered a concern. However, if you intend to use the display module for developing multiple projects, then you may want control the use of the flash memory as the program destination.

It should be noted also that if the device is unplugged after a program is downloaded to RAM, the program would then be lost when the device is powered up again. A program downloaded to the flash memory, on the other hand, would still be present after a power cycle. This is because RAM is a volatile memory while flash memory is a non-volatile memory. Thus, if the program is to be retained after a power cycle, then it must be downloaded to the flash memory.

Lastly, note that the option of choosing RAM as the program destination is available only for Picaso and Diablo16 display modules. The default and only program destination option for Goldelox display modules is the flash memory since the Goldelox processor has only 510 bytes of RAM.

Execution of a Program Downloaded to RAM

Picaso Processor

A program downloaded to RAM resides in the RAM section and it will execute from there. When a program is downloaded to the RAM section of a Picaso processor, Workshop will print the information below in the message area.

```
No Errors, code size = 42 bytes out of 14400 total
Initial RAM size = 200 bytes out of 14400 total
Program will run from ram so total initial RAM
size = 242 bytes out of 14400 total
Download to RAM successful.
```

In the information shown above, the size of the program is 42 bytes and it would occupy **42** bytes out of the **14400** bytes of available Picaso flash memory space if it were finally placed in the flash memory.

Now a program might require a certain amount of RAM to run. For the above example, the program requires **200** bytes out of the **14400** bytes of available Picaso RAM space to run. Now since the 42-byte program was downloaded to and resides in the RAM section and since it needs an

additional 200 bytes of RAM space to execute, the total RAM requirements of the application is therefore **242** bytes out of the **14400** bytes of available Picaso RAM space.

Lastly, note that the example above uses no flash memory at all since the program was downloaded to RAM. When the display module is power-cycled, the program in the RAM would be lost and the program previously downloaded to the flash memory, if any, would execute.

Diablo16 Processor

When a program is downloaded to the RAM section of a Diablo16 processor, Workshop will print the information below in the message area.

```
No Errors, code size = 42 bytes out of 32750 total
Initial RAM size = 200 bytes out of 32768 total
Program will run from ram so total initial RAM
size = 242 bytes out of 32768 total
Download to RAM successful.
```

In the information shown above, the size of the program is 42 bytes and it would occupy **42** bytes out of the **32750** bytes of available Diablo16 flash memory space if it were finally placed in the flash memory.

The program requires **200** bytes out of the **32768** bytes of available Diablo16 RAM space to run. Now since the 42-byte program was downloaded to and resides in the RAM and since it needs an additional 200 bytes of RAM space to execute, the total RAM requirements of the application is therefore **242** bytes out of the **32768** bytes of available Diablo16 RAM space.

Execution of a Program Downloaded to Flash

Run from RAM

So after having been tested and finalized, the program is now ready to be downloaded to the flash memory. Note that, by default, a program downloaded to the flash memory will, when the processor starts running, be first copied to RAM and will be executed from RAM. Consider the information shown below. This information is printed by Workshop in the message area after a program is downloaded to the flash memory of a Picaso processor.

```
No Errors, code size = 42 bytes out of 14400 total
Initial RAM size = 200 bytes out of 14400 total
Program will run from ram so total initial RAM size
= 242 bytes out of 14400 total
Download to Flash successful.
```

First, note that the last line is now “Download to **Flash** successful”. Thus, the 42-byte program now resides in the flash memory. Note that the program needs 200 bytes of RAM to run. Since the processor, during start up, creates a copy of the program and places it in the RAM section, the total RAM requirements of the application is 242 bytes.

Shown below is the printed information when the same 42-byte program is downloaded to the flash memory of a Diablo16 processor.

```
No Errors, code size = 42 bytes out of 32750 total
Initial RAM size = 200 bytes out of 32768 total
Program will run from ram so total initial RAM size
= 242 bytes out of 32768 total
Download to Flash successful.
```

Shown below is the printed information when the same 42-byte program is downloaded to a Goldelox processor.

```
No Errors, code size = 42 bytes out of 9216 total
Initial RAM size = 0 bytes out of 510 total
Download successful.
```

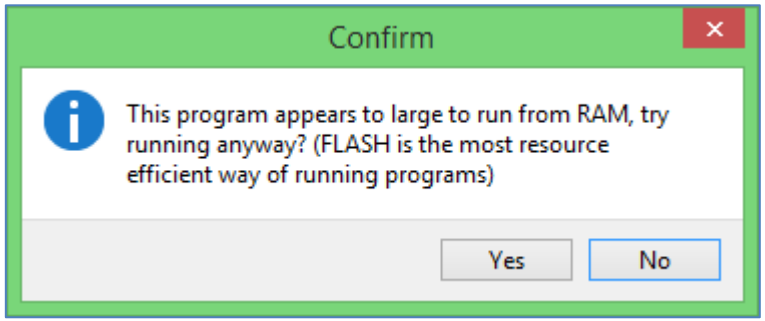
Again, note that the flash memory is the only available destination for a Goldelox program.

Run from Flash

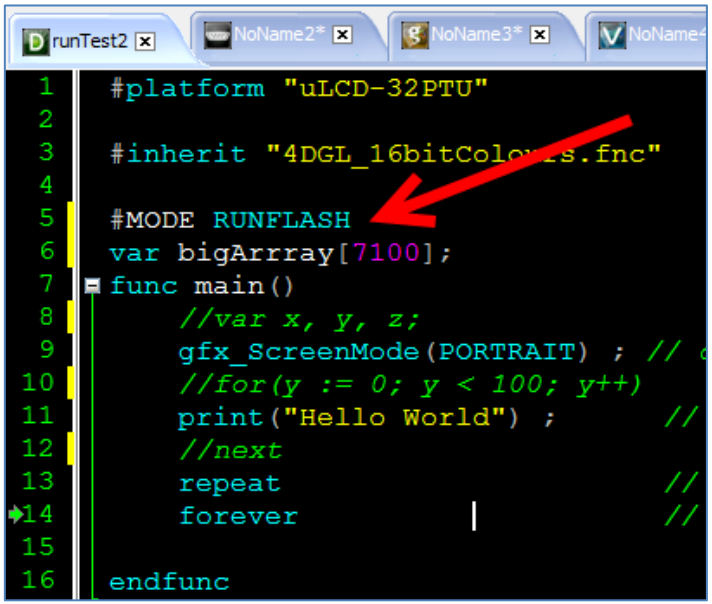
Besides running from RAM, a program downloaded to the flash memory can also run from the flash memory (where it resides). This way, the processor will not need to create a copy of the program and place it in the RAM section. To explain the need for the run-from-flash option, we suppose that the 42-byte Picaso program in our previous example requires 14400 bytes of RAM space to run. Now since the Picaso RAM size is only 14400 bytes, we cannot use the default run-from-RAM option, where the processor creates a copy of the program and places it in the RAM. Doing so would not be possible since the total RAM requirements would be 14442 bytes, which is larger than the available RAM of the Picaso processor. In the event that you have accidently compiled the program just recently described, Workshop would print the information shown below.

```
No Errors, code size = 42 bytes out of 14400 total
Initial RAM size = 14400 bytes out of 14400 total
Program will run from ram so total initial RAM size
= 14442 bytes out of 14400 total
Insert #MODE RUNFLASH and select Destination FLASH
to run program from FLASH
```

Workshop would also show a dialog box.



The correct course of action in this case is to click No and insert the pre-processor directive "**#MODE RUNFLASH**" in to the beginning part of the code, like as shown below.



The pre-processor directive "**#MODE RUNFLASH**" would make the program run from the flash memory, where it resides. Thus, no RAM space would be needed to hold the program. Workshop would then print the information shown below.

```
No Errors, code size = 42 bytes out of 14400 total
Initial RAM size = 14400 bytes out of 14400 total
Download to Flash successful.
```

Although the above example is for the Picaso processor, the concept applies to the Diablo16 processor as well. Note however that the pre-processor directive "#MODE RUNFLASH" does not apply to Goldelox processors. Also, using "#MODE RUNFLASH" in Diablo16 and Picaso programs downloaded to RAM wouldn't work, since programs downloaded to RAM will reside in the RAM and will execute from there.

Run from RAM vs Run from Flash

So we have seen that the run-from-flash option can be used to conserve RAM space. Note however that, for the Picaso processor, a program running from the flash memory is significantly slower compared to that which is running from RAM. For most uses however, this difference is not that noticeable. For the Diablo16 processor, a program running from RAM has about the same speed with that which is running from the flash memory.

In other words, after downloading a Picaso program to the flash memory and using the run-from-flash option, you might, depending on the nature of your project, notice a slightly slower performance as compared to that when you were testing it on RAM (download to RAM) or when you were using the default download-to-flash-and-run-from-RAM option. Again, for most cases, however, this difference in performance is not noticeable.

Saving Programs to the uSD Card

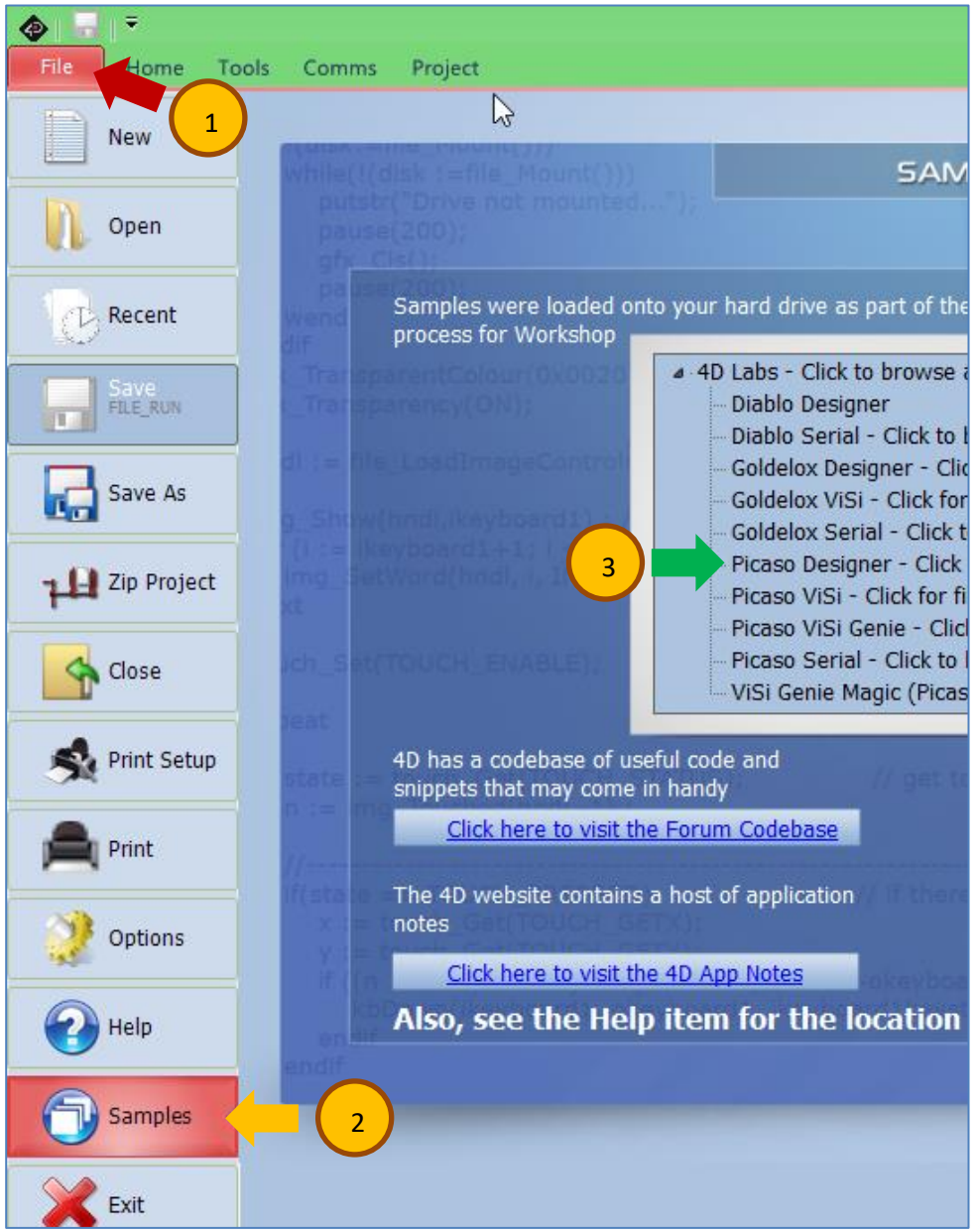
There is a third destination option for programs – the uSD card. This scenario involves the use of parent and child programs. To accomplish this, the first step is to compile a program and save it to the uSD card. This is the child program. The second step is to create another program and download it to the display module's processor. This is the parent program. During runtime, the parent program will access the uSD card, look for the child program, and load it to RAM. The child program will then execute. Below are the three FAT16 file functions used in loading child programs to RAM.

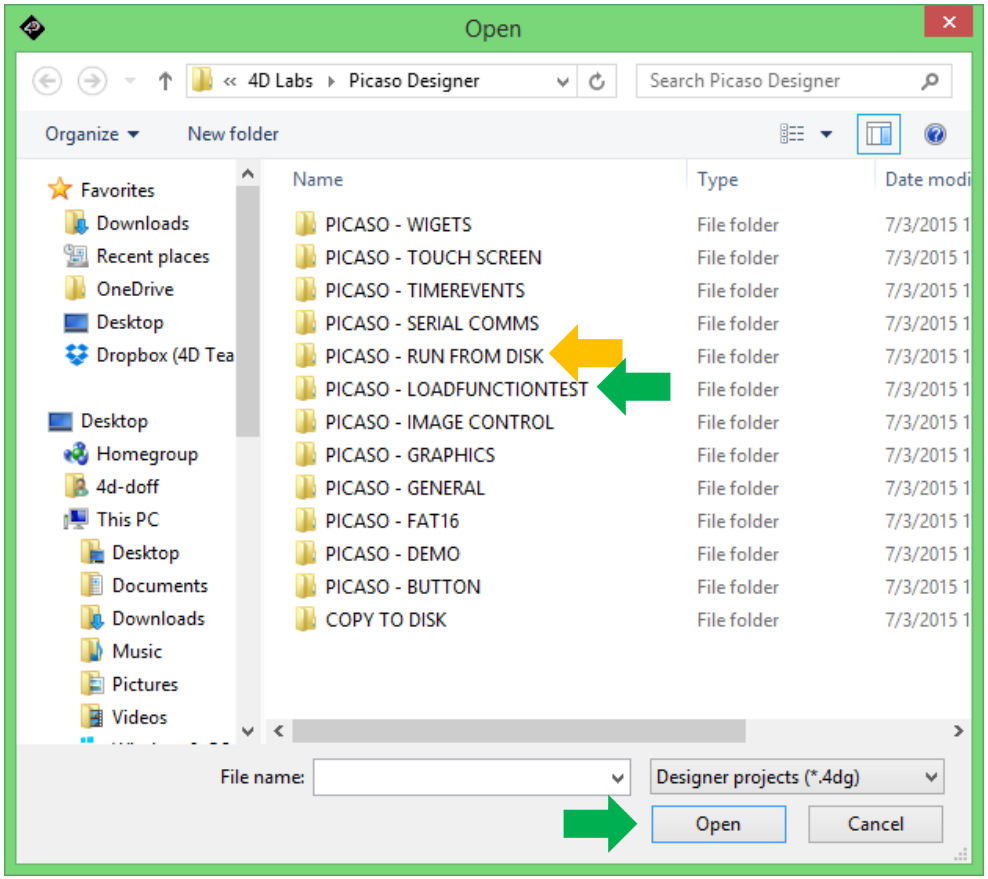
```
file_LoadFunction(...)
file_Run(...)
file_Exec(...)
```

For more information on these functions, refer to the Picaso and/or Diablo16 Internal Functions Manuals.

A large program can be divided into smaller parts, which are then saved as separate child programs. Using a parent program, any of the child programs can then be loaded to RAM and executed as required by the application. When a child program is no longer needed, it can be discarded and the memory allocated to it can be released. Therefore, with proper memory management, any project requiring a memory space larger than the allotted RAM (14KB for Picaso and 32KB for Diablo16) and flash memory (14KB for Picaso and 32KB per bank for Diablo16) to run can be broken down and

implemented using a parent program and several child programs. The Workshop IDE comes with numerous examples of projects using parent and child programs. See the image below for directions. Although the examples were tested on a Picaso display module, they should run normally on a Diablo16 display module.




Open
« 4D Labs ▸ Picaso Designer
Search Picaso Designer
Organize
New folder
Favorites
Downloads
Recent places
OneDrive
Desktop
Dropbox (4D Tea
Desktop
Homegroup
4d-doff
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Name
Type
Date modi
PICASO - WIGETS
PICASO - TOUCH SCREEN
PICASO - TIMEREVENTS
PICASO - SERIAL COMMS
PICASO - RUN FROM DISK
PICASO - LOADFUNCTIONTEST
PICASO - IMAGE CONTROL
PICASO - GRAPHICS
PICASO - GENERAL
PICASO - FAT16
PICASO - DEMO
PICASO - BUTTON
COPY TO DISK
File folder
7/3/2015 1
File name:
Designer projects (*.4dg)
Open
Cancel

.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability