What is DigitaScript?

Digita Script is a programming language that runs on a Digita enabled digital camera (see Table 1 for a listing of Digita enabled cameras and compatible version of DigitaScript). It allows better access to camera functions and capibilties of the camera that are only available though DigitaScript.

Examples of its use include automation of image capture, watermarking images with information from other devices, HTML code generation, mulltiple custom camera setups, and guided capture. Numerous scripts written by Digita camera owners are available for free on the Internet.

Manufacturer	Camera Name	Script Version
Kodak	DC 220 Zoom	DigitaScript Version 1.0
	DC 260 Zoom	DigitaScript Version 1.0
	DC 265 Zoom	DigitaScript Version 1.1
	DC 290 Zoom	DigitaScript Version 1.5
Minolta	Dimage Ex Zoom 1500 Dimage Ex Wide 1500	DigitaScript Version 1.1
	Dimage Ex Zoom 1500 Ver 2 Dimage Ex Wide 1500 Ver 2	DigitaScript Version 1.1
Hewlett Packard	PhotoSmart C500	DigitaScript Version 1.5

Table 1. - Digita Enabled Cameras

Digita scripting language is easy to learn and use.

What Can DigitaScripts Do?

Scripts are powerful tools that help you get the most from your digital camera. Scripts have been developed for numerous functions. For example, there are scripts which:

- identify the type of picture to be taken (portrait, sports, action, etc.) so the camera can record the best possible image
- automate setting the text, color, position, date, time, and logo for a watemark
- set white balance and retrive exposure information

Scripts can be used for business or hobby pursuits, as shown by these examples.

Business Use

Let's say a real estate agent would like to post a listing on the Internet. Using the wrong pictures with a listing or forgetting to take key shots can be frustrating. To keep this from happening, a script can be developed to assist the agent with obtaining, categorizing, and uploading the necessary pictures to a PC or website. A DigitaScript program can even generate the HTML file right in the camera so it is ready for upload.

Hobby Use

Many people who take photographs as a hobby strive to take professional looking photographs. To get professional results, one technique is to take multiple shots of the same scene with different exposures and focus settings to get the “perfect” picture. If you create a DigitaScript program to do this, it is possible to take such a sequence of photographs by pressing the shutter only once.

Do you have difficulties in finding the camera settings and changing them? With a script, you can save multiple configurations and set them from one menu.

Taking pictures of fireworks can be difficult because you might need to set multiple camera settings. Once you have figured out the best setting, you can write a script that will automatically set all of the settings. Next time you want to take pictures of fireworks, just run your “Fireworks Photo Setting” script and you're ready to go!

If you go fishing and catch the big one, you'll want to take a photo, right? What about the location of your big catch? If you attach a GPS receiver to your camera and a script that talks to it, it is possible to attach the exact location of your big catch into the image.

Writing a DigitaScript Program

To write scripts you will need the following items:

- A simple text editor (like WordPad or NotePad)
- A Digita enabled camera

Note: *A Script Reference manual is available from FlashPoint Technology Inc.'s web site (http://www.flashpoint.com). This manual is an in-depth look at the conventions and syntax of DigitaScripts. It also describes the commands and parameters necessary for programming DigitaScripts.*

The Development Process

When you develop scripts, it is recommended that you follow the five steps of the development process:

- Planning/Investigating
- Creating/Editing
- Transferring to Camera
- Testing
- Completion

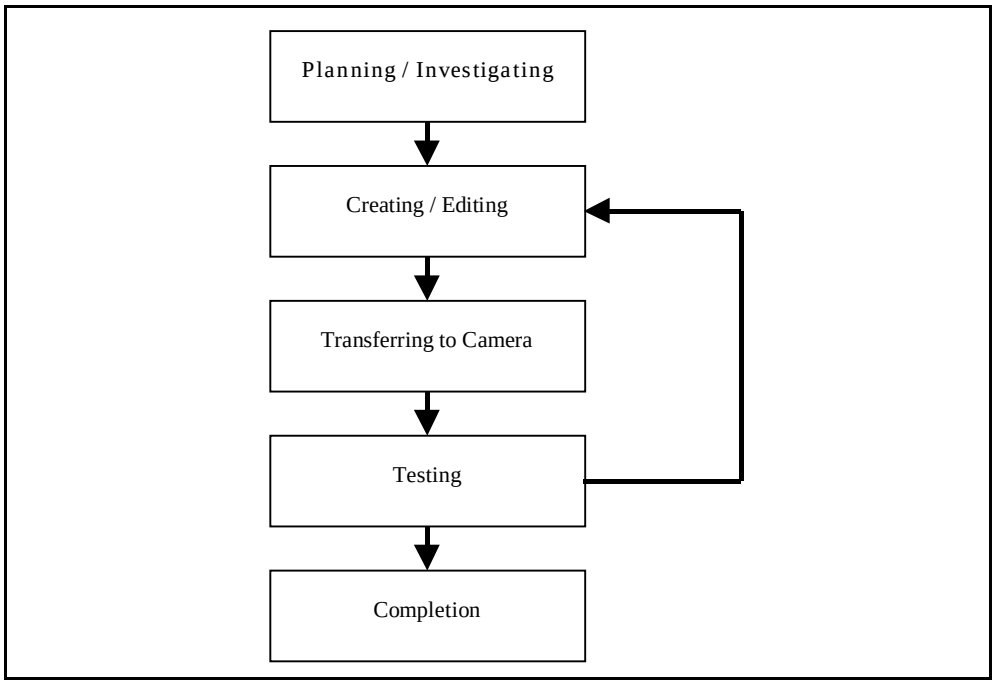


Figure 1. Development Process

Planning/Investigating Stage

The planning/investigating stage is the most important stage since it will dictate what you will do in your script. In this stage, you need to visualize the solution and decide which parameters and commands to use.

In order to create a good solution, think of “What is the problem I am trying to solve?” Clearly articulate the problem, then think about “What is the ideal solution to the problem?”

When you understand and have a clear picture of the problem and solution in your mind, you can proceed to the creating/editing stage.

Creating/Editing Stage

You can use any text editor to write your script program. Be aware that the DigitaScript Language is *case sensitive*. Many programming errors are due to misspelled commands and improper use of case in variable names.

Also keep in mind that all DigitaScript programs require the name, mode, menu, and label commands to be declared. The mode declaration determines which mode the script will run in (Capture, Review, or Play). The menu declaration determines which menu the script will be listed under on the camera. The label is the text that appears in the menu listing.

Figure 2 shows a DigitaScript program with the necessary commands highlighted.

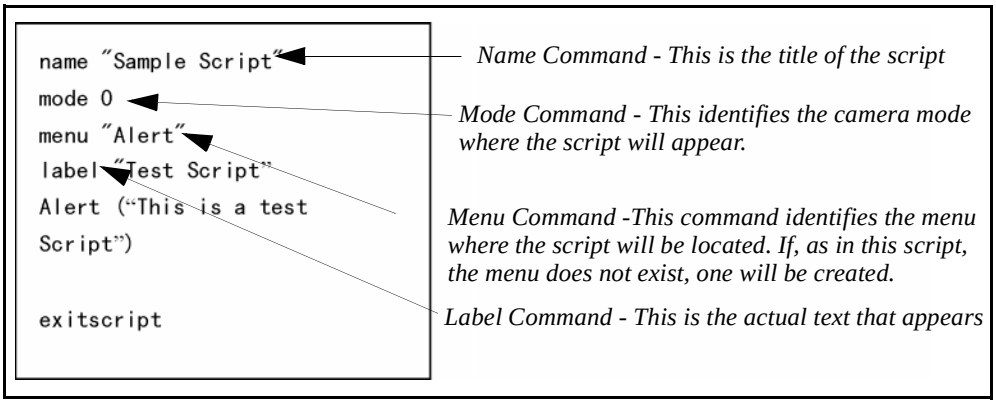


Figure 2. Sample DigitaScript

When you are done writing your script, save the file with a "`.csm`" extension at the end. For example, `SampleScript.csm`.

Note: *Script names should comply with the 8.3 DOS standard naming conventions, otherwise the camera will shorten the name.*

Transferring to Camera

To run the script program, you need to copy the program/script file you created to your digital camera. There are 2 ways to transfer your scripts to the camera 1) with a CompactFlash card reader, or 2) by using additional software.

CompactFlash Card Reader

If your computer is equipped with a CompactFlash card reader, directly copy the script into the System folder of the CompactFlash card. If no system folder exists, you will need to create one. Figure 3 shows the System file located inside the CompactFlash card (drive F).

Note: *In Figure 3, the CompactFlash card has been assigned to F drive. Your drive designation will depend on your computer configuration.*

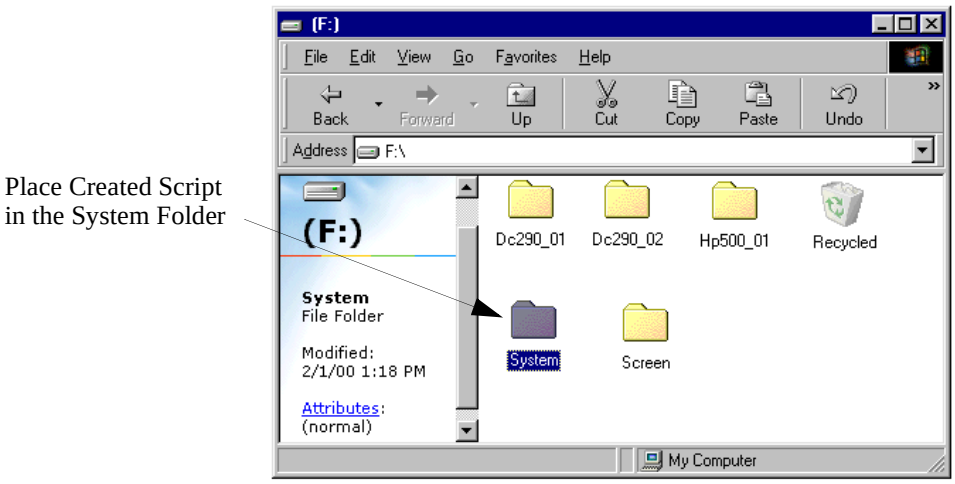


Figure 3. System Folder Location

Additional Software

You can use a software program (like Kodak Mounter or Digita Desktop) to transfer data to the camera. This method is explained in the instruction manual that came with the camera or the software program. Refer to the manual(s) for further instructions.

Testing

The title of the script will be displayed in the Menu Display of either the Capture/Record or Review, or Play mode. You can choose which mode you want your script to run in by the mode command in the script (in Figure 2 Capture mode was used). The options are as follows:

- 0 - Script displayed in Capture/Record mode
- 1 - Script displayed in Review mode
- 2 - Script displayed in Play mode

Note: *Some mode values are not available in some cameras.*

If the script shown in Figure 2 was transferred to a camera, a new menu entitled **Alert** would be created in the camera's capture mode as displayed in Figure 4.

Figure 4. Alert Menu Created from Script

If you highlight **Test Script** and press the **Start** softkey the information you entered under the label command will be displayed as shown in Figure 5.

Figure 5. Test Script

While this is a very simple script, the essential commands (name, mode, menu, and label) will function the same regardless of a script's complexity.

Note: *For more detailed information on DigitaScript command and parameters refer to the Script Reference manual.*

Debugging

The following are some hints for efficient script program debugging:

- While testing, it is a good idea to have your computer next to you. When an error occurs, the line number where the error occurred at will be displayed so you can confirm and revise your script.
- If there is no response from the camera, the program maybe in an infinite loop (which means the script executes the same series of instructions over and over again). When this happens, the only way to stop your script program is by removing the power to the camera. To remove power to the camera, follow these steps:
  - Make sure that the CompactFlash card is not being accessed before powering down. (The lamp next to the card slot will be flashing if the card is being accessed.) Remove the A/C adapter or the batteries if the power does not turn off.
  - If the card slot light does not turn off, wait about a minute. If the light does not turn off, take the batteries out. When this happens, the data on the card may be corrupt so you may need to reformat the card. If you are forced to do this, it's always a good idea to check the CompactFlash card for errors with a utility like ScanDisk or Norton Disk Doctor. When the camera is rebooted a message will be displayed stating that the card needs to be reformatted.

Note: Refer to the camera's instruction guide if you need to format the card.

- Because the LCD consumes a lot of battery power, it is recommended that you use the A/C power adapter when you are developing and testing scripts.

Completion

When you are done, save a copy of your script on your computer. If you reformat your CompactFlash card, the script will be erased.

Comments

It is also recommend that you add your name and the program creation date as comments at the beginning of the program. The script will not execute any commands in the comment section. Comment lines being with a “#” character.

You may also want to add copyright information and a contact e-mail address if you plan to distribute your script.

Figure 6 shows an example of comments at the beginning of the program.

```
#/*****************************************/
# Camera Setting for Fireworks
#
# Copyright 1999/11/05 Hiroshi Igami
# WebPage http://www.cityfujisawa.ne.jp/h_igami
#/*****************************************/
```

Figure 6. Script Comments

Writing your first DigitaScript program

Let's write a simple script by following the development process. The script we are going to make is an old-time programming sample called “Hello World”. Your finished script will display “Hello World” on the camera screen for 15 seconds.

Planning/Investigating

The program flow should be:

1. Display “Hello World”
2. Wait 15 seconds

In this stage, you need to look at the Script Reference manual to find whether there are commands to do steps 1 and 2. For now, by looking in Appendix B of this guide, you will find the Display and Wait commands and their functions which are shown in Table 2.

Command	Function
Wait	Specifies the number of milliseconds for the script program to wait.
Display	Places feedback on the LCD.

Table 2. - Wait and Display Commands

Creating/Editing

Now that we know the commands that we are going to use in our program, it is time to start creating the script.

Open a Text Editor such as WordPad or NotePad. Add some comments and the necessary commands such as those shown in Figure 7.

```
#/*********************************/
#  A Script Program
#  Displaying Hello World
#  Copyright 1999/11/13 Hiroshi Igami
#/*********************************/
name "Hello World Program"
mode  0
menu  "Sample Scripts"
label "Hello World"

exitscript
```

Figure 7. DigitaScript Comments and Commands

As a quick reminder, here are the *required* DigitaScript commands:

- `name` - Name of your script
- `mode` - Which mode to run in
- `menu` - Which menu the script will be displayed in
- `label` - What to display in the menu label

When you are done entering any comments and the required commands, add the Display and Wait commands as shown in Figure 8.

```
#/*********************************/
#  A Script Program
#  Displaying Hello World
#  Copyright 1999/11/13 Hiroshi Igami
#/*********************************/
name "Hello World Program"
mode  0
menu  "Sample Scripts"
label "Hello World"

# Display
Display("Hello World")
Wait( 15000 )

exitscript
```

Figure 8. Display and Wait Commands Added to Hello World Script

DigitaScript commands use characters or numbers as arguments. Notice that the `Hello World` argument in the `Display` command is enclosed within quotation marks (`"`). This

designates this argument as *literal text* which means it will be displayed on the LCD. The `Wait` command requires a number in milliseconds (1/1000 seconds). To wait 15 seconds, you need to enter (`15000`).

Note: *Detailed explanations of these commands are in the Script Reference manual.*

Save the script as "`Hello.csm`".

Transfer the Script to the Camera and Test

Copy the script into the camera's System folder.

`Hello.csm` sets the script to run in mode 0 which is the Capture/Record mode. Power on the camera and go to Capture/Record mode. Press the menu and go to the Sample Scripts menu. Your Script should appear under this menu as "Hello World" as shown in Figure 9.

Figure 9. Hello World Script

This is the script you have just created. Now, test it by highlighting "Hello World" and pressing the **Start** softkey.

You will see "Hello World" on your camera screen for 15 seconds (Figure 10).

Figure 10. Hello World Script Message

Note: *If an error occurs, (substituting uppercase for lowercase letter in a command, i.e., using "display" instead of "Display".)) an error message will be displayed on the LCD. This message will display the line number where the error occurred and a brief explanation of the error.*

Completion

Save your scripts on a PC when it is completed because your program will be erased if your CompactFlash card is reformatted.

DigitaScript Programming Elements

In this section, we will explain the programming elements of DigitaScript language and suggest some methods that may help you create better scripts.

Variables

For those of you who are not familiar with any programming languages, a "variable" might be a confusing term. Think of a variable as a container to store numbers or characters. Using this container, the program can add/subtract numbers or make decisions based on what is stored in them.

DigitaScript language supports the variables shown in Table 3.

Data Type	Description	Version
u	An unsigned integer (example: 1, 2, 17, etc.)	1.0/1.1
i	A signed integer (example: -1, -2, -17, etc.)	1.0/1.1
f	A floating point number (example: 2.3, -9.7)	1.0/1.1
s	A string with a maximum length of 32	1.0/1.1
n	A 8.3 character DOS file name (example: LScripts.csm)	1.0/1.1
b	Boolean 1 (true) or 0 (false)	1.0/1.1
t	A string with a maximum length of 255	1.5

Table 3. DigitaScript Supported Variables

It is very important for you to know the characteristics of the variables in Table 3. For example, if you want to store the number “1.5”, you will need to use a variable with the data type “f” for floating point number.

If you are reading characters from the serial port, you will use type “s” or type “t”. If the characters are more than 32 characters in length, you will need to use type “t” or characters will get truncated.

Using Variables

Variables must be declared before they are used. By declaring variables, you are telling the DigitaScript interpreter what variables will be used. The script interpreter will then allocate a “container” for the declared variables. Varialbles can only be declared once in a script or a syntax error will occur. The value of the variable can be set as many times as needed.

If you want to make your script easy to understand, use descriptive veriables. Some suguestions on variables:

- Always start with the data type (u, i, f, s, etc.), and something descriptive.
- When setting and retrieving values add a capitals S (set) or R (retrieve/returned) for the second character of the variable. This technique works well for Get and Set commands.

  Examples: `uAValue`, `uBValue`, `uResultValue`, `sWatermarkText`, `uSshut`, `uRshut`.

The sample script in Figure 11 adds two numbers. This script declares three variables, *uAValue*, *uBValue*, and *uResultValue*, as unsigned integers. When you are declaring more than one variable on one command line, use a comma to separate them.

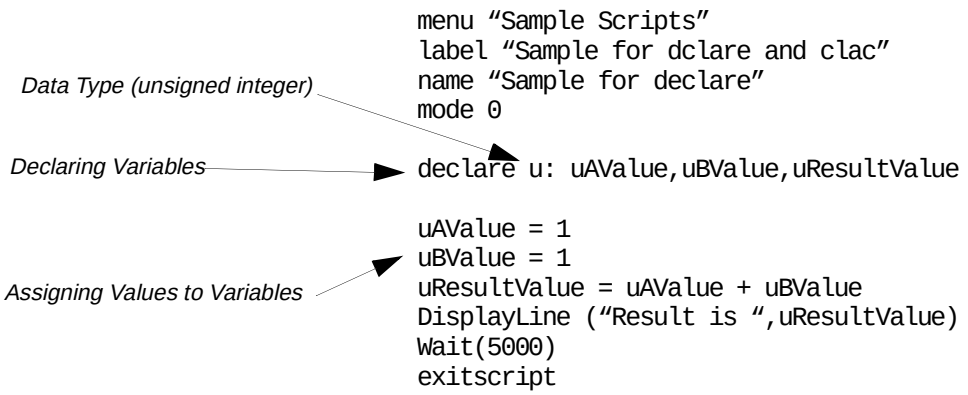


Figure 11. Sample Script to Add Two Numbers

Note: *In the sample in Figure 11, the variable names begin with the actual data type of the variable (u). By doing this, you know exactly what data type the variable is by looking at the name. This will be helpful when you are trying to find error in your script.*

Parameters

Parameters are used to set camera values to specific settings. DigitaScript uses the following types of parameters:

- *Product Information Parameters* - Allows you to make inquiries concerning the camera or firmware being used. These parameters are read only.
- *Camera Capabilities Parameters* - These parameters give you more control over your camera and its picture taking abilities. For example, parameters of this type let you specify the interval between picture captures (`tldy`) and set the image size for timelapse captures (`tsiz`).
- *Image File Tags* - These parameters provide information about the pictures you are taking. For example, parameters of this type provide information about the level of image compression (`cmpn`) and image capture date (`date`).

To demonstrate the use of parameters, consider taking pictures of fireworks. When you are doing this, you may want to increase exposure time to 6 seconds and set the focus to infinity. The parameters `xmod`, `shut`, `fmod`, and `fdst` can be used to accomplish this, as shown in the script snippet in Figure 12.

```
# Set to shutter priority mode
SetCameraState("xmod", 2)

# Set shutter speed to 6 seconds
SetCameraState("shut", 6000000)

# Set to manual focus mode
SetCameraState("fmod", 3)

# Set focus distance to infinity
SetCameraState("fdst", 65535)
```

Figure 12. Fireworks Script Snippet

All the parameters used in DigitaScripts are covered in detail in the Script Reference manual. Each parameter has a table like the one shown below for `xmod`. As detailed in the table, when `xmod` is set to 2 (as in the syntax in Figure 12) the exposure mode becomes shutter priority. By setting it in this mode and then selecting a long shutter speed a longer exposure time is set.

xmod– Exposure Method

Definition: Specifies the exposure mode for the camera.
Data Type: Enum List
Access: Read/Write

Associated File Tag: xmod

Product/OS	Details	Factory Default	Stored In
Digita Script	1=Auto, 2=Shutter Priority, 3=Aperture Priority, 4=Gain Priority, 5=Programmed, 6=Manual		EEPROM
Kodak DC 260/DC 265DC 290	2 =long exposure mode, 3=external flash sync mode, Settings 4, 5 and 6 not supported.	1	EEPROM
Kodak DC 220	2 =long exposure mode, Settings 3, 4, 5 and 6 not supported.	1	EEPROM
Dimage EX 1500	Settings 2, 5 and 6 not supported.	1	EEPROM
HP PhotoSmart C500	1=On, Settings 2, 3, 4, 5, and 6 not supported.	1	EEPROM

Arithmetic Operation

DigitaScript language is capable of adding, subtracting, multiplying, and dividing numbers. Writing these operations is fairly straightforward as shown in the example in Figure 13.

```
declare u: ua, ub, ures

ua = 4
ub = 2
#  addition
ures = ua + ub
#  subtraction
ures = ua - ub
#  multiplication(*)
ures = ua * ub
#  division (/)
ures = ua / ub
```

Figure 13. Arithmetic Operations

Conditional And Branch Statements

DigitaScript language includes conditional branching (if-end statements) as well as unconditional go-to statements.

if ~ end statement

If - end conditional branches will evaluate the statement following the "`if`" syntax. If the statement is true, then the script will continue to execute the next line until the end is reached. Conditional statements are comparisons that either can be true or false. Depending on the result of the comparison, you can make the script do different things.

The syntax displayed in Figure 14 will display "`It is 1!`" on the camera if the variable *utest* is "1".

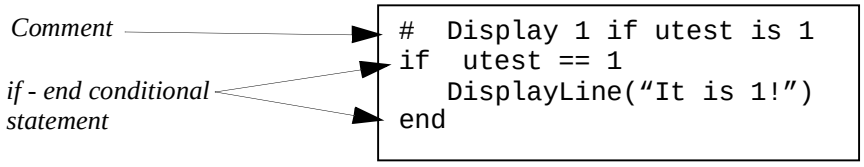


Figure 14. if - end Conditional Statement

Table 4 shows a list of applicable operators that can be used when comparing for the condition.

Operators	Definition	Valid Data Types
>	Greater than	u, i, f, b
>=	Greater than or equal to	u, i, f, b
<	Less than	u, i, f, b
<=	Less than or equal to	u, i, f, b
==	Equal to	u, i, f, s, n, b, t
!=	Not equal to	u, i, f, s, n, b, t
~	One's complement	u, i, b
&	AND	u, i, b
|	Inclusive OR	u, i, b
^	Exclusive OR	u, i, b

Table 4. Applicable Operators

Goto and Reentry statements

This command allows a script to jump from one point and reenter into another part of the program. Figure 15 shows the script jumping to a point called "`Reentry:`" and displaying "`2`" instead of "`1`".

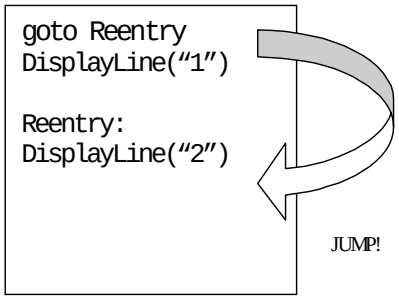


Figure 15. Example of Goto Statement

ExitScript Statement

The exitscript statement can be placed anywhere in the script to terminate execution of the script program.

Independent and Grouped Commands

Within the Digita language there are independent and grouped commands.

Independent Commands

An independent command can stand on its own. An example of this type of command would be `StartCapture( )`. Most of the Digita commands are in this form. You do not need to worry about initializing and cleaning up after the command.

Grouped Commands

Grouped commands need to be used in a particular sequence to accomplish the intended task. Consider this example of grouped commands to display an option list:

```
SetOption
GetOption
```

The `SetOption` command establishes the return values and items in an option list, and the `GetOption` command displays the list to the user for input. The important concept in the above example of grouped commands is that for this script to work as intended, `SetOption` must execute before `GetOption`.

Two more examples that demonstrate the importance of grouped command sequences are shown below.

Example 1

```
SerialOpen
SerialSendReceive/SerialSend/SerialReceive
SerialClose
```

Example 2

```
FileOpen
Write/WriteLine/Read
FileClose
```

Example 1 (`SerialOpen`) opens the serial port at the selected data transfer rate (baud rate) and Example 2 (`FileOpen`) opens a file with the specified file name.

Generally, there is a sequence in commands that involve interaction with an external device. You may include other commands between each of the above commands, but the sequence is important. If you do not “Open” before trying to access, you will get an error.

Note that it is also important to “Close” what you have “Opened”. For example, if you forget the `FileClose` command, the file may get corrupted or get strange results in another part of the program by writing to the wrong file unintentionally.

Note: *Both independent and grouped commands are covered extensively in the Script Reference manual.*

Examples of DigitaScripts

This section presents some short DigitaScript examples that give common programming techniques.

Note: More advanced examples are displayed in Appendix D.

Command Repetition

Generally, when you need to issue the same command repetitively, the "`goto`" command is used.

Look at the script in Figure 16. It is a simple script that calculates the sum of the numbers 1 through 5. First, the variables "`ucounter`" and "`usum`" are defined. We will use "`usum`" to store the result. This script will execute the lines between "`Reent`:" and the "`goto Reent`" five times. When "`ucounter`" is greater than 6, "`usum`" will be displayed.

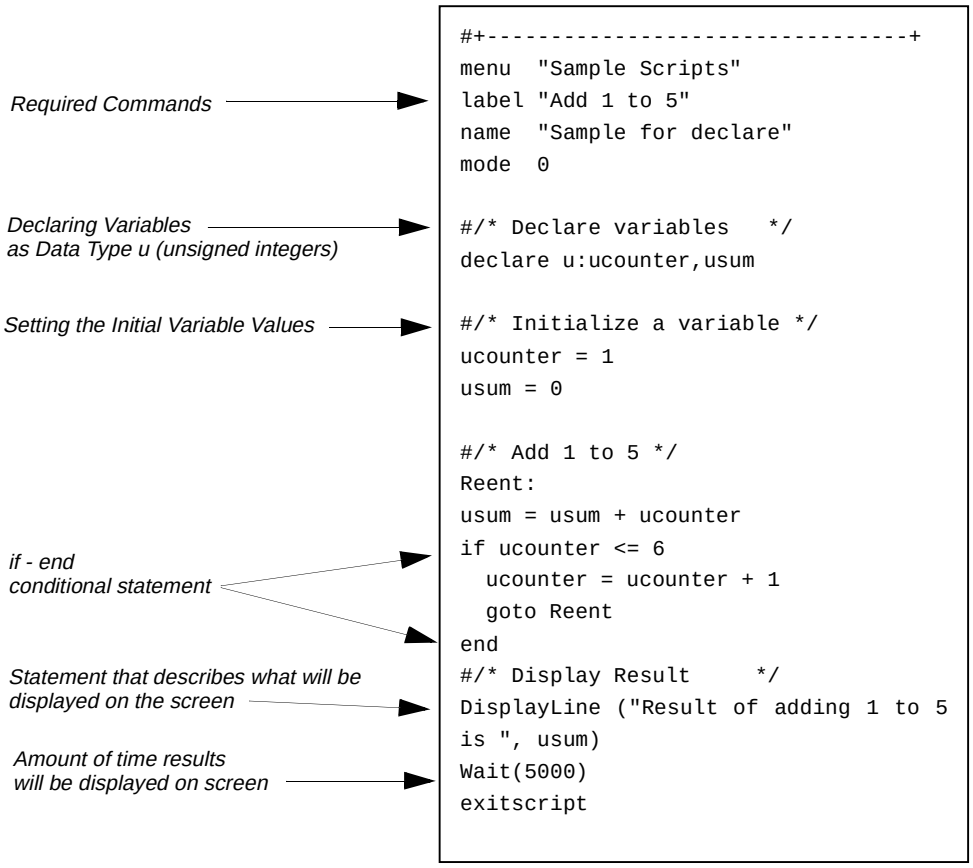


Figure 16. Command Repetition Script

Displaying an Option List

An Option list allows you to receive input from a user using the camera's softkeys. To use this command, you must first define the option list by using the command `SetOption`. Then you can use the `GetOption` command to get the input from the camera user. Figure 17 shows the script used to create an option list with three possible selections (Select 1, Select 2, and End).

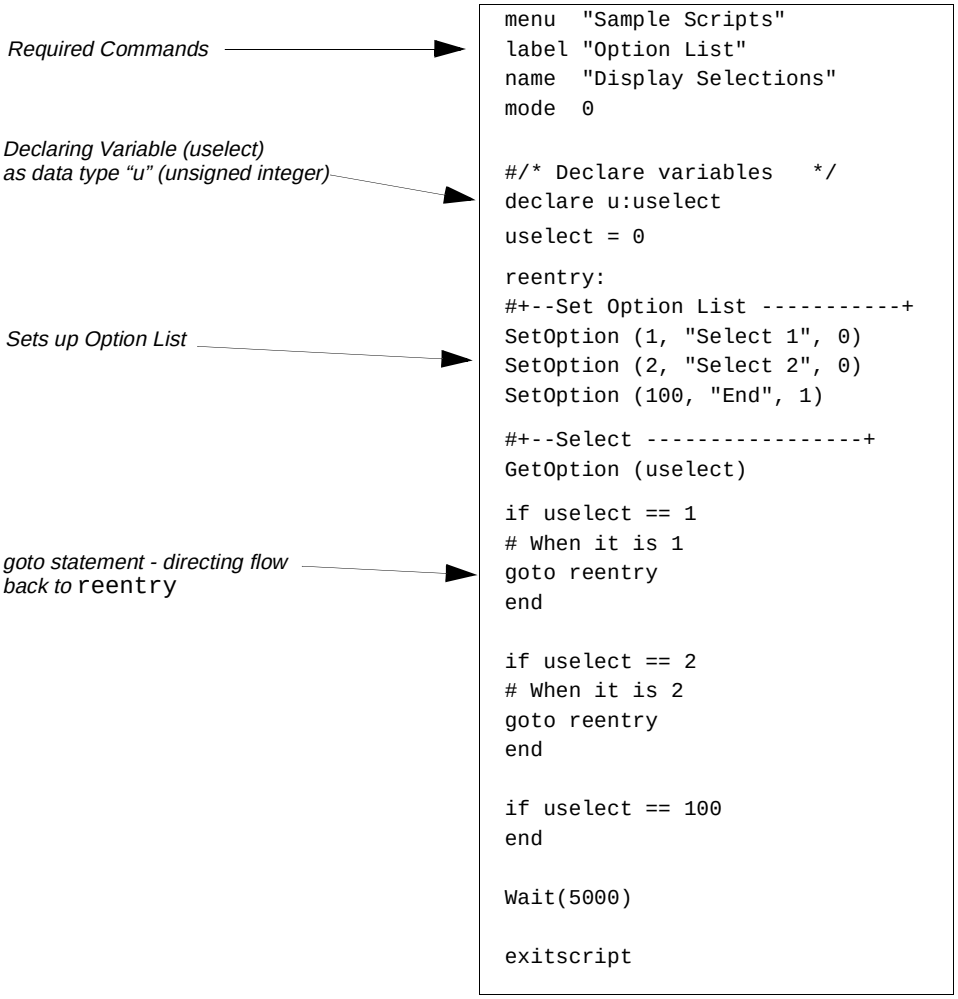


Figure 17. Example Script for Displaying an Option List

If the script in Figure 17 is installed in the system folder on your FlashCard, you will see the Option List script displayed under Sample Scripts when the camera is in the capture mode (Figure 18).

Figure 18. Sample Script Menu, Option List Script Included

To start the Option List press the Start softkey. Figure 19 displays the next screen that will appear based on the script created in Figure 17.

Figure 19. Option List Script Started

Monitoring the Camera

The need could arise to monitor if the camera is ready to take another picture in instances where successive pictures are to be taken. To do this, a script can be developed to find out the status of the camera, using the "`GetCameraStatus`" command.

This command returns three bit flags, `SystemStatus`, `CaptureStatus`, and `VendorStatus` that contain various status data from the camera. `GetCameraStatus` is used to poll the camera on a regular basis, and the bit flags indicate whether or not the internal state of the camera has changed.

Note: *A bit flag is a 32 bit (b) type data structure with the data starting at the most significant bit. For more information on bit flags and bit definition refer to the Script Reference manual.*

The following syntax, when inserted into an existing script, will ensure the script will check to see if the camera is ready to take more pictures.

```
#/***Examining camera check loop ***/

WaitWorking:
status = 0
GetCameraStatus (bSys, bCap, bVen)
if bSys & 0x10000000
  Wait (500)
  goto WaitWorking
end
```

Start-up Scripts

It is possible to write a script that performs actions for you automatically every time the camera is powered on. For example, you might want to move the zoom lens to a certain position, set a special white balance value, or over-ride the camera default values that cannot be saved through a power cycle.

To have a script run every time the camera is powered on, name it `startup.csm` and put it in the system folder of your Compact Flash Card.

When you are developing a start-up script, you should develop the script using another file name, make sure that it does exactly what you want, then finally rename it to `startup.csm`. This will help avoid instances where the camera will start up with an error because of problems in the start-up file.

Scripts that Check the Camera Model

When you are writing scripts that are dependent on a camera feature, you should ensure the script checks for the camera model. For example, if your camera does not have a 3x zoom capability, you will not be able to configure it for 3x zoom.

Your script can determine the camera model by the "`GetProductInfo`" command and the "`ptid`" parameter.

The script syntax displayed in Figure 20 will check to see if the camera is a Kodak DC290 Zoom. If the camera is another version or brand, the message "**This camera is not a DC290 Zoom**" will appear and the script will stop running.

```
declare      s: sProdname

#/***Check to see if it is DC290 or not ***/
GetProductInfo ("ptid", sProdname)
if sProdname != "KODAK DC290 Zoom Digital Camera"
    DisplayLine (" This camera is not a DC290 Zoom!")
    exitscript
end
```

Figure 20. Script Syntax to Check Camera Make and Model

Table 5 lists the Digita supported cameras and their ptid value.

Camera Type	ptid
Kodak DC290	KODAK DC290 Zoom Digital Camera
Kodak DC265	KODAK DC265 ZOOM DIGITAL CAMERA
Kodak DC260	KODAK DIGITAL SCIENCE DC260
Kodak DC220	KODAK DIGITAL SCIENCE DC220
Minolta Dimage EX1500	Dimage EX

Table 5.Camera ptids

Appendix A: Programming Basics

Style

There have been many books written and every programmer has an opinion on how programs should be formatted.

However, it is good practice to use basic formatting techniques to make the program easier to understand and debug. Some of these techniques that are especially for DigitaScript programming are described below:

Indentation

Most samples have indentation after the beginning "`if`" statements. Indentation makes it easier to understand what gets executed if the statement is true.

Declare all of your variables near the beginning of the script

This will help prevent you from using the wrong variables or declaring it more than once.

Use capital letters for the goto label

If all letters are capitalized, it is easy to find where the program has jumped to. Using a descriptive label also helps. Examples: `MAIN`, `LOOP`, `FLASH`, `ON`.

Conditional Statements and Loops

Any time you write a script that could end up in loop, you should add a `Wait` command. This will allow you to exit the script without removing power.

Note: For information on the `Wait` *command, refer to the Scripting Reference Guide.*

A Review of Base Accounting

Decimal numbers (everyday counting numbers) are based on 10 digits, binary numbers are based on two digits and hexadecimal numbers are based on 16 digits. Consider the following three ways to express the decimal number 26:

In base 10, the breakdown would be as follows:

Power of 10	10^1	10^0	
Decimal Value	10	1	
Decimal Number	2	6	This equates to 2*10 + 6*1 = 26

In binary form, 26 would be expressed as 11010. This breaks down as

Power of 2	2^4	2^3	2^2	2^1	2^0	
Decimal Value	16	8	4	2	1	
Binary Number	1	1	0	1	0	This equates to 16*1 + 8*1 + 2*1 = 26

In hexadecimal form 26 is expressed as 1A. In hexadecimal numbers, there are only nine numerals that represent 1-9; 10-15 are represented by A-F respectively. The following shows the hexadecimal breakdown for 26.

Power of 16	16^1	16^0	
Decimal Value	16	1	
Hexadecimal Number	16	A	This equates to 16*1 + 10(A=10)*1 = 26

Hexadecimal numbers are described in more detail below.

Base 16 (Hexadecimal) Numbers

The computer only understands numbers in zeros and ones, which is called the binary numbering system. This can be very confusing and complex. One way to reduce the complexity is to represent numbers in hexadecimal notation. To signify that the number is hexadecimal, it is common to put a “0x” in front of the number to say that it is in HEX notation.

From the following table, you can see that 15 is F in hexadecimal and 1111 in binary. This is convenient since the computer usually handles these binary numbers eight at a time. The eight zeros or ones, called bits, is referred to as a byte.

Decimal Number	Hexadecimal Representation	Binary Representation
0	0x0	0000
1	0x1	0001
2	0x2	0010
3	0x3	0011
4	0x4	0100
5	0x5	0101
6	0x6	0110
7	0x7	0111
8	0x8	1000
9	0x9	1001

10	0xA	1010
11	0xB	1011
12	0xC	1100
13	0xD	1101
14	0xE	1110
15	0xF	1111

Table 6.- Comparison of Decimal, Hexadecimal, and Binary Numbers

Now lets look at 0xFF. A chart comparing the hexadecimal and the binary representation of this number would look like the following:

Hexadecimal	**F**				**F**			
Binary	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

From the chart above, you can see that the binary representation for 0xFF is 11111111. For data that is four bytes, which is 32 bits long, the same rule still applies. Break up into chunks of four bits and find the hexadecimal number for each four bits.

Appendix B: Script command summary

Command	Description
Product and Image Information Commands	
GetProductInfo	Requests specific information about the product by parameter.
GetImageSpecifications	Returns hardware-related image information, including zone organization.
Status Commands	
GetCameraStatus	Returns system status, capture status, and vendor status.
GetError	Clears error flag and returns last error and its description.
Option List Commands	
SetOption	Displays an option list to the user.
GetOption	Prompts the user for an option selection.
Camera Capabilities and State Commands	
GetCapabilityType	Returns capability type associated with the specified camera parameter.
GetCapabilitiesRange	Returns the min-max range information associated with the specified camera parameter.
GetCapabilitiesCount	Returns the number of capability list items defined for the specific camera parameter.
GetCapabilitiesListItem	Returns a name-value pair for the specified camera parameter's list index.
GetCapabilitiesValue	Returns the capability value defined for the specified camera parameter.
GetCameraState	Requests a current camera parameter setting.
SetCameraState	Updates a current camera parameter setting.
GetCameraDefault	Requests the user or factory default value of a camera parameter setting.
SetCameraDefault	Updates the user default value for a camera parameter setting.
RestoreCameraDefault	Restores either the user defaults or the factory defaults.
Power and Capture Commands	
GetPowerMode	Determines the power level available to the camera.
SetPowerMode	Powers off the camera.
SetCaptureMode	Controls the type of capture sequence: still, group, or timelapse.
StartCapture	Starts the capture process.
EndCapture	Stops the capture process.

File Commands	
GetFileCount	Returns the total number of captured images resident on the disk.
GetFileInfo	Accesses the list of all captured image files currently available.
GetNewFileCount	Returns the number of new images resident on the disk.
GetNewFileInfo	Accesses the list of all newly created image files currently available.
EraseFile	Delete image files or other files.
GetStorageStatus	Determines the status of storage available to the camera.
GetFileTag	Returns the value of the specified file tag.
MakeFolder	Creates a new directory
SetUserFileTag	Sets the value of the specified user tag.
Date and Time Commands	
GetClock	Gets the current date and time.
SetClock	Sets the current clock value for date and time.
GetDateString	Returns the date as a formatted string.
GetTimeString	Returns the time as a formatted string.
Camera Script Execution Commands	
GetSystemFileCount	Returns the number of system files resident in the SYSTEM folder.
GetSystemFileName	Returns a single system file name based on an index.
RunApp	Restarts the camera using the selected application.
RunScript	Runs a script file that is listed in the SystemFilesList.
GetScriptName	Retrieves the long name of a script
Serial Processing Commands	
SerialOpen	Opens the serial channel at the selected baud rate.
SerialSendReceive	Sets up for the next data exchange through the serial port.
SerialSend	Sends the indicated data and sets up the next receive cycle.
SerialReceive	Waits to receive an expected transmission.
SerialClose	Releases the serial port.
File and Read/Write Commands	
FileOpen	Opens a file in the specified directory with the specified file name.
FileClose	Closes a file.
Read	Reads data from a text file.
ReadLine	Reads one line of data from a text file.
Set Delimiter	Sets the delimiter used by the Read and ReadLine commands.

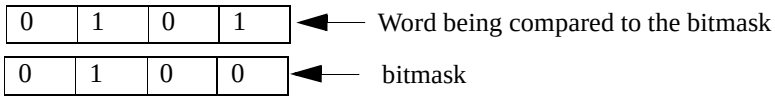
Write	Appends the specified data to the current pointer location in an open file (no carriage return added at end).
WriteLine	Appends the specified data to the current pointer location in an open file and adds a carriage return to the end of the data.
String Manipulation Commands	
FindString	Places the pointer at the start of a substring within a source string.
NumberToString	Converts a numerical value to a string value.
StringToNumber	Converts a string to a numerical value.
Substring	Extracts and returns a substring from a source string.
Wait and Display Commands	
Wait	Specifies the number of milliseconds for the script program to wait.
Display	Places feedback text on the LCD.
DisplayLine	Places feed back text on the LCD and appends a carriage return.
DisplayClear	Clears the LCD.
GetString	Requests an input text string from the user.
WaitForShutter	Returns camera system control to the user while waiting for the shutter button to be pressed. Valid only in "capture" mode.
Alert	Places an alert on the LCD.
Image Commands	
GetMarkedImageCount	Returns the number of images currently marked.
GetMarkedImage	Retrieves an image file.
MarkImage	Marks a specified image.
MarkAllImages	Marks all the images on the CompactFlash card.
UnmarkImage	Unmarks a specified image.
UnmarkAllImages	Unmarks all the images on the CompactFlash card.

Appendix C: Bitmasking

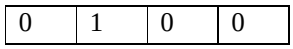
Bitmasking is not limited to DigitaScripts but is performed in many computers. The way you find out the status of things is to read the status not a bit at a time but at bytes at a time. The only way to get at each bit in your data is to compare it with a bit mask using the AND(&), OR(|), XOR(^) operations. What do each of these symbols mean?

AND Operation (&)

When the both are True (1), the result is True(1)



When you AND the above numbers, the result will be:

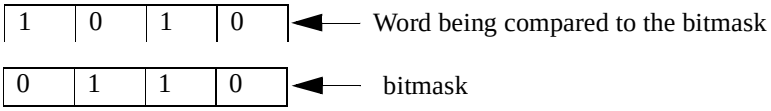


This result comes about because the second column is the only column where there are two true (1) bytes.

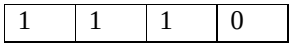
OR Operation (|)

When either is True (1), the result will be True.

Put a one where you want ones.



When we OR the above two numbers, we get:

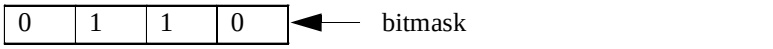


This result comes about because the fourth column is the only column where there are two false (0) bytes.

XOR(Exclusive OR) Operation (^)

This is the opposite of OR. It is most commonly used to set a bit that is true (1) to false (0).

Put a one where you want to make the bytes false (0).



When XOR the numbers above, we get:

1	1	0	0

Monitoring if the Camera can take another Picture

As an example of bitmasking, let's consider a script that will monitor if the camera is ready to take another picture in an instance when successive pictures need to be taken. To do this, the script can find out the status of the camera using the "`GetCameraStatus`" command.

This command returns three bit flags, `SystemStatus`, `CaptureStatus`, and `VendorStatus` that contain various status data from the camera. Get `CameraStatus` is used to poll the camera on a regular basis, and the bit flags indicate whether or not the internal state of the camera has changed. In the following syntax, an `if` condition has been set to check if the camera's system status is ready to take another picture.

```
#/***Examining camera check loop ***/

WaitWorking:
status = 0
GetCameraStatus (bSys, bCap, bVen)
if bSys & 0x10000000
  Wait (500)
  goto WaitWorking
end
```

`if` condition to check camera's system status

A bit flag, such as `SystemStatus`, is a 32 bit "b" (unsigned 32-bit binary integer) type data structure with the data starting at the most significant bit. The bit flag for `SystemStatus` is shown below.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ed1a	Ed2a	ramda	ipip	Memf	pwra	flsc	trun	stst	stsc	cerr	Mcro	Sbs1	Sbs2	zoom	extp	edch	rsvd														

To find the value of the specific bit, you need to compare with a bit mask. For example, if you want to find the value of `ipip` (Image Processing In Progress - the 28th flag in the `SystemStaus` data structure, shown above), you would need to compare it with a bit mask, which means you perform an AND operation between the bit flag and the hexadecimal number 0x10000000 (0x10000000 is specified in the script).

***Note**: The 0x at the beginning signifies a number as being hexadecimal. For a review of Base Counting refer to Appendix A. In hexadecimal form, 2^{28} is expressed as 0x10000000.*

When the an image is being processed the `SystemStatus` bit flag would look like the following (notice that the `ipip` flag is toggled):

0	0	0	1	0	0		0	0	0
31	30	29	28	27	26		2	1	0
			ipip						

If you AND the above bitflag with the bitmask of 0x 10000000 in binary form shown below (recall that $0x10000000=16^7=(2^4)^7=2^{28}$).

0	0	0	1	0	0		0	0	0
31	30	29	28	27	26		2	1	0

Then the result will also have a true (1) setting in the 28th bit position. Therefore the `if` statement is true and so the script will wait the required time (500 seconds) before taking another picture.

When the camera is finished processing the image the `SystemStatus` bit flag will change to show a false (0) bit for the `ipip` flag.

0	0	0	0	0	0		0	0	0
31	30	29	28	27	26		2	1	0
			ipip						

When an AND operation is performed with the bitflag shown above and the one for 0x10000000 the result will be zero. Therefore the `if` statement is false and the script goes on to take the next picture.

Appendix D: Advanced DigitaScript Examples

Happy Wedding

This Script is an example of how to control the zoom function with a DigitaScript program. You can use this at wedding receptions and parties to take pictures automatically. The difference between a normal timelapse capture is that it will change zoom positions between shots. When you use this script, you should mount the camera on a tripod and use an AC power adaptor.

```
#/**************************************/
# Happy Wedding
#
# Copyright 1999/11/30 Hiroshi Igami
#/**************************************/

name "Happy Wedding"
mode  0
menu  "Sample Scripts"
label "Happy Wedding"

declare i:istatus , itelezoom , iwidezoom,iimgraw
declare u:uipip , unowside, ustore, uitaken, uiavail
declare u:uwait
declare b:bsys, bcap, bven

uipip = 0x10000000
unowside = 0

#/*script startup messege */
DisplayLine ("Auto Photo Script")
Wait (1500)
```

Mask bit data to check if the camera is ready to capture another picture.

Continued…

In the above section, variables are declared and initialized. Then a short message "**Auto Photo Script**" is displayed on the LCD screen.

```
#/*********************************************/
# Prepare to capture
#/*********************************************/

#/*Set capture to still mode*/
SetCaptureMode (still)

#/*Capture interval time options*/
SetOption (0, "Minimum", 0)
SetOption (30000, "30 seconds", 0)
SetOption (60000, "1 min", 0)
SetOption (120000, "2 min", 0)
SetOption (300000, "5 min", 0)
#+--Get time interval-----------+
GetOption (uwait)
DisplayClear()


#/*Get zoom setting for telephoto */
WaitForShutter("Zoom in and Press Shutter")
GetCameraState ( "zpos", itelezoom )
EndCapture ()

#/*Get zoom setting for wide */
WaitForShutter("Zoom out and Press Shutter")
GetCameraState ( "zpos", iwidezoom )
EndCapture ()

#/*Set to wide */
SetCameraState ( "zpos",  iwidezoom )
```

Continued…

In this section, the capture interval is set. Then the user can choose two arbitrary zoom positions. This is useful because the best framing is dependent on the room size and where the main activity will be.

```
#/***********************************************/
# Auto Capture Processing
#/***********************************************/
Pictakeloop:
#/*Check to see if there is enough space to store more images or
not*/
GetStorageStatus (1, uitaken, uiavail, iimgraw)
if uiavail < 1
  DisplayClear()
  DisplayLine("Memory full")
  Wait (1500)
  goto Fin
end

#/*** Capture ***/
SetCaptureMode (still)
StartCapture ()
EndCapture ()

#/***Check to see if saving is completed or not*/
Waitproc:
istatus = 0
GetCameraStatus (bsys, bcap, bven)
if bsys & uipip
  Wait (1500)
  goto Waitproc
end

#/***Wait a bit before taking the next picture***/
DisplayClear()
DisplayLine("Waiting")
Wait (uwait)

#/***Change zoom setting
if unowside == 0
   unowside = 1
   SetCameraState ( "zpos", itelezoom )
   goto Pictakeloop
end
if unowside == 1
   unowside = 0
   SetCameraState ( "zpos", iwidezoom )
   goto Pictakeloop
end
Fin:
exitscript
```

In this section, the pictures are captured until the card is full.

House Guide

This script is an example of a guided capture. The script asks the user to take specific pictures and then generates a HTML file with the pictures.

Before taking the picture, be sure to input the group/album name in the camera.

```
#/*************************************/
# House Guide
#
# Copyright 1999/12/01 Hiroshi Igami
#/*************************************/

name "House guide"
mode  0
menu  "Sample Scripts"
label "House guide"

#/* Define variables */
declare u: ufd ,ufd2,ufd3, ucount,umore
declare u:uipip,uflsc,ulength,icounter
declare s: spath ,schkpath,starget
declare n: nname,nfirstname
declare b:bsys, bcap, bven,bstatus
declare u:uscene
declare s:sphotopos,sphotomsg

#/* Initialize fixed value
uipip = 0x10000000
uflsc = 0x00800000
```

Continued...

This is the initialization and declaration section. The variables `uipip` and `uflsc` are obtained from the "`GetCameraStatus`" command. They are initialized because they are needed to set the bitmask for use in conditional statements.

The next section goes through the process of capturing the first image. Then it waits until the file is processed.

```
#/*************************************************/
# Check
#/*************************************************/
#/*Set capture mode to still mode */
SetCaptureMode (still)
WaitForShutter("Take the Outside")
StartCapture ()

#/*************************************************/
# Check file list to see if processing is finished
#/*************************************************/
Waitproc:
GetCameraStatus (bsys, bcap, bven)
if bsys & uipip
  Wait (1500)
  goto Waitproc
end
Waitflsc:
GetCameraStatus (bsys, bcap, bven)
if bsys & uflsc
  Wait (1500)
  goto Waitflsc
end
```

Continued…

In the next part of the script, the "`GetNewFileCount`" and "`GetNewFileInfo`" commands will read all album/group names. This script will check to see if the name is already being used with the "`ChkLoop`" command.

```
#/************************************************/
# Check group name from file list
#/************************************************/
GetNewFileCount (ucount)
GetNewFileInfo (0,spath,nfirstname,ulength,bstatus)
EndCapture ()

#/************************************************/
# Check to see if group is already in use or not
#/************************************************/
Chkloop:
  GetFileInfo (0,schkpath,nname,ulength,bstatus)
  if spath == schkpath
     DisplayClear()
     DisplayLine("Picture already exists in the Album/Group.")
     DisplayLine("Make   a   new   album/group   or   delete   the
picture.")
     EraseFile (spath, nfirstname)
     Wait (5000)
     exitscript
  end
  icounter = icounter +1
  if icounter <ucount-1
    goto Chkloop
  end
Startshot:
```

Continued…

The next section of the script uses the "`FileOpen`" command. The file name used in the `FileOpen` command needs to be a full file path. We have all the strings but need to join or concatenate the strings together. However, there is no way to concatenate strings in Digita scripting language. One way to solve this problem is to create a temporary file and to write out the concatenated string then to read from it. Here we use this trick to concatenate the full file path name.

```
#/************************************************/
# Create file name
# Because concatenation of strings is not possible
# we write out as file
#/  input  : spath
#/  output : starget
#/************************************************/
EraseFile ("/", "worktemp")
FileOpen (2, "worktemp", 1, ufd)
Write (ufd, "/")
Write (ufd, spath)
Write (ufd, "index.htm")
FileClose (ufd)
FileOpen (2, "worktemp", 2, ufd2)
ReadLine (ufd2, starget)
FileClose (ufd2)


#/************************************************/
# Capture and create HTML
#/************************************************/
EraseFile (spath, "index.htm")
```

Continued…

The last part of the script (on the next page) creates a HTML file and the links to the first picture. The rest of the sequence will also capture the rest of the pictures. Photographs of the entrance, kitchen, and dining room are also programmed in.

```
#/*************************************************/
# Create header and first page of the HTML file
#/*************************************************/
FileOpen (2, starget, 1, ufd3)
WriteLine (ufd3, "<HTML>")
WriteLine (ufd3, "<HEAD>")
WriteLine (ufd3, "<Title> House Guide</Title>")
WriteLine (ufd3, "</HEAD>")
WriteLine (ufd3, "<BODY>")
WriteLine (ufd3, "<TABLE BORDER>")
WriteLine (ufd3, "<TR>")
WriteLine (ufd3, "<TD>")
Write (ufd3, "Album Path :")
WriteLine (ufd3, "<TD>")
Write (ufd3, spath)
WriteLine (ufd3, "<TR>")
WriteLine (ufd3, "<TD>")
WriteLine (ufd3, "Outside")
WriteLine (ufd3, "<TD>")
Write (ufd3, "<IMG SRC=\"")
Write (ufd3, nfirstname)
Write (ufd3, "\"   WIDTH=320>")
```

```
#/*************************************************/
# Capture specified place
#/*************************************************/
uscene = 0
sphotopos ="Entrance"
sphotomsg ="Take Entrance"

#/***** Capture loop     ****/
Photoloop:
SetCaptureMode (still)
WaitForShutter(sphotomsg)
StartCapture ()
Waitproc2:
GetCameraStatus (bsys, bcap, bven)
if bsys & uipip
  Wait (1500)
  goto Waitproc2
end
Waitflsc2:
GetCameraStatus (bsys, bcap, bven)
if bsys & uflsc
  Wait (1500)
  goto Waitflsc2
end
GetNewFileCount (ucount)
GetNewFileInfo (0,spath,nname,ulength,bstatus)
EndCapture ()
WriteLine (ufd3, "<TR>")
WriteLine (ufd3, "<TD>")
WriteLine (ufd3, sphotopos)
WriteLine (ufd3, "<TD>")
Write (ufd3, "<IMG SRC=\"")
Write (ufd3, nname)
Write (ufd3, "\"  WIDTH=320>")
WriteLine (ufd3, "<BR>")
WriteLine (ufd3, "<TD>")

#/*** Set capture scene(add below) **/
if uscene == 0
   uscene = 1
   sphotopos ="Kitchen"
   sphotomsg ="Take Kitchen"
   goto Photoloop
end
if uscene == 1
   uscene = 2
   sphotopos ="Dining"
   sphotomsg ="Take Dining"
   goto Photoloop
end
```

Continued…

```
#/*****************************************************************/
# Additional capture
#/*****************************************************************/
MorePhoto:
DisplayClear()
SetOption (0, "More Pictures", 1)
SetOption (1, "Finish Taking", 0)
GetOption (umore)

if   umore == 0
   SetCaptureMode (still)
   WaitForShutter("More Photos")
   StartCapture ()
  Waitproc3:
   GetCameraStatusb(sys,  bcap, bven)
  if    bsys & uipip
    Wait (1500)
      goto Waitproc3
  end
  Waitflsc3:
   GetCameraStatusb(sys,  bcap, bven)
  if    bsys & uflsc
    Wait (1500)
      goto Waitflsc3
  end
   GetNewFileCount (jcount)
   GetNewFileInfo (0, spath, nname, ulength, bstatus)
   EndCapture ()
   WriteLine (ufd3, "<TR>")
   WriteLine (ufd3, "<TD>Others")
   WriteLine (ufd3, "<TD>")
  Write (ufd3, "<IMG SRC=\"")
  Write (ufd3,    nname)
  Write (ufd3, "\"  WIDTH=320>")
   WriteLine (ufd3, "<BR>")
   WriteLine (ufd3, "<TD>")
   goto MorePhoto
end
WriteLine (ufd3, "</TABLE>")
WriteLine (ufd3, "</BODY>")
FileClose (ufd3)

#/*****************************************************************/
# End
#/*****************************************************************/
DisplayLine ("Finished")
exitscript
```

Language Changing Script

This is an example of a script that will check your camera's language capabilities and then allow you to change them to your preference. Following the syntax are camera screen shots displaying the actions of the script.

```
name "Language Settings"
mode 0
menu "Preferences"
label "Language..."

declare u:rgnc, count, default, uIndex, value
declare s:desc

GetCapabilitiesCount("rgnc", count, default)
GetCameraDefault(1, "rgnc", rgnc)

uIndex = 0

SetUpOptions:

if count == uIndex
goto Finish
end

GetCapabilitiesListItem("rgnc", uIndex, desc, value)

SetOption(value, desc, rgnc == value)

uIndex = uIndex + 1

goto SetUpOptions

Finish:

GetOption ( rgnc )
SetCameraState ("rgnc",rgnc)
SetCameraDefault (2,"rgnc",rgnc)
exitscript
```

This section of the script checks the camrea's capabilities and then displays them (See Figure 21)

This section allows you to choose and set a region (language) and retain it as the default (See Figure 22).

Figure 21. Language Script Displayed under Preferences Menu

Figure 22. Available Languages Displayed

Figure 23. Japanese Selected as Language Preference and Set as Default

In this guide, we introduced several sample scripts, but there are more available from Digita sites such as `http://www.DigitaPhoto.com` and `http://www.digitacamera.com`.