Script Guide

Digita™ Operating Environment version 1.5

Part Number 6-2140-0001-01

Version 1.5

November 18, 1999

Version 1.5
November 18, 1999

Table of Contents

About This Book

Purpose

This manual provides an introduction to the Digita™ Script language, including an overview of the language and various techniques used in writing scripts.

Intended Audience

This manual is intended for users who have some prior experience with a scripting or programming language. It does not teach basic programming techniques.

Related Manuals

The *Digita Script Reference* provides a complete description of the Digita Script language.

Text Conventions

`code`	sample code
italics	references to other manuals

Chapter 1 Digita Script Language Overview

Digita™ script is a series of commands, expressed in plain ASCII text, which can be interpreted by system software in a Digita-equipped camera. The scripts are independent of the actual Digita camera. As scripts are loaded, the camera menus display the available scripts. Users select scripts from the menu options through the camera control buttons. The script language provides commands to request additional user input during script execution.

The script interpreter processes the script line by line. The script language includes conditional branching ("if" constructs) as well as unconditional go-to statements. In addition to these basics, the language provides specialized commands to control the camera, retrieve information from the camera, and access the image data on disk. Also, the serial port can be used to control remote devices.

The Digita script language supports marker and go-to control statements, keywords, common data types (signed and unsigned integer, bitwise Boolean, fixed point, and strings) numeric and logical operators, declaration statements, and variables and constants for all data types.

Scripts can be factory-installed in camera ROM or stored in removable media mounted on the camera. Factory-installed scripts are always available for use, while scripts on removable media are available only when the media is inserted in the camera. Examples of scripts include:

- Scripts that provide hints to the camera about the type of picture to be taken (portrait, sports action, and so on) so it can record the best image possible.
- Scripts that set white balance, bracketing, and other exposure information.
- Scripts that set up image stamps to place watermarks on the image. Typical image stamps are date and time, company logos, and signatures.
- Scripts that prompt users to take certain pictures. These scripts are useful for users who must document scenes according to a prescribed protocol. Examples of such users are:
  - Insurance claim adjusters, who must document accident scenes or insured property.
  - Real estate agents, who need a standardized series of images of property for sale, perhaps for uploading to a World Wide Web site.
  - Police photographers, who must document evidence from crime scenes.

Writing and Using Scripts

Scripts are plain text files created on Macintosh®, Windows®, or Unix® systems. The names of the files follow DOS® naming conventions; that is, they are of the form XXXXXXXX.CSM. The first eight characters can be any combination of letters, numbers, and underscore characters (_). The last four characters must be .CSM in order for Digita to recognize the file as a script. Scripts are placed in the SYSTEM directory on the camera, usually on a CompactFlash card. Each file name must be unique.

To install a script, simply place it in the SYSTEM directory of the camera. To do so, copy the file to the SYSTEM directory from a computer or via a connection utility such as Digita Desktop™. Please see the user manual for the application or the computer operating system for more information on copying files. The Digita operating environment recognizes that a new script has been installed when the camera is restarted or when the CompactFlash card is removed and reinserted.

To actually use a script, enter menu mode on the camera. Then, select the script from whichever menu it appears under. Press the softkey labeled “Start” (Minolta users press the “Edit” softtkey) to run the script.

Scripts have several statements which are required. A script begins with a name statement, followed by mode, menu, and label statements. After the label statment comes the body of the script. The last line of the script is an exitscript statement. A brief explanation of each of these statements is found below. Detailed explanations are found in the *Digita Script Reference*.

Script Structure

Scripts have several statements which are required. A script begins with a name statement, followed by mode, menu, and label statements. After the label statement comes the body of the script. The last line of the script is an exitscript statement. A brief explanation of each of these statements is found below. Detailed explanations are found in the *Digita Script Reference*.

Name Statement

The name statement is a long name for the script. The name is a string of up to 31 characters. It must be enclosed in double quotes (""). This name is used primarily from within other scripts via the GetScriptName() command. It does not appear in menus.

The format of the name statement is:

```
name "name of script"
```

This example sets the name of the script to "name of script".

Mode Statement

The mode statement is used to tell the camera in which mode to display the script in a menu. The mode is a single number which specifies the mode. Currently, the supported modes are 0 (capture), 1 (review), and 2 (play). Note that the review and play modes have been combined on the Kodak DC220, DC260, DC265, and DC290, and are the same for scripting purposes.

The format of the mode statement is:

```
mode 0
```

This example sets the mode for the script to the capture mode.

Menu Statement

The menu statement tells the camera under which menu to display the script label. The menu name is a string of up to 31 characters enclosed in double quotes (""). If the menu does not exist in the specified mode, it is created. If the menu does exist, the camera places the script under that menu after the last item in the menu.

The format of the menu statement is:

```
menu "menu name"
```

This example sets the name of the menu in which this script appears to "menu name".

Label Statement

The label statement tells the camera what name to display for the script under the specified menu. The label consists of up to 31 characters enclosed in double quotes (""). Usually, they will be fairly short in order to make them more readable. If a script uses the same label as another script in the same menu, both labels will appear under the menu. As this is confusing to users, it is useful to label each of your scripts differently.

The format of the label statement is:

```
label "label name"
```

This example sets the name which appears on the menu specified for this script to "label name".

Exitscript Statement

The last statement in a script should be an exitscript statement. It tells the script interpreter to close the script and return the camera to the standard operating mode.

The format of the exitscript statement is:

```
exitscript
```

An exitscript statement may also be placed elsewhere in a script. For example, the display could show a menu that asks the user whether or not they want to capture more images. If they don’t, the script can execute an exitscript statement to terminate the script without requiring a jump to the end of the script.

Sample Script

Below is an example of the minimum elements necessary for a valid script. Note that this script does nothing except appear in the capture mode in the Sample Scripts menu with the label Do Nothing.

```
name "Do nothing script example"
mode 0
menu "Sample Scripts"
label "Do Nothing"
exitscript
```

Other Statements

The rest of the script between the label and exitscript statements consists of a series of statements. These statements tell the camera to actually do something, such as get information from the user, write information to a file, or display information to the user. The other statement types are covered below. Details on these statements are available in the *Digita Script Reference*.

Comment Statements

Single-line comment statements start with the # symbol. The # symbol may be followed by any number of characters. Comments are used to give information to anyone reading the script. They are often used to give details about the name of the script, how it operates, and how specific parts of the script work. Comments stop at the end of the line on which the # symbol appears; that is, after a carriage return. To continue a comment statement on another line, the other line must also begin with a # symbol.

The format of the single-line comment statement is:

```
# Some text which makes up the comment
```

Multi-line comment statements start with /* and end with */. Any number of characters and carriage returns can be entered in between the multi-line comment indicators. Multi-line comments can only be used in scripts written specifically for products compatible with Digita Script version 1.5 or newer.

The format of the multi-line comment statement is:

```
/* Some text
which makes up
the comment */
```

When there are many comments throughout a script, it is much easier to understand the logic of the script, both for scripts written by oneself and especially for scripts that may be maintained or used by others.

Declaration Statements

Declaration statements allow you to tell the script interpreter what identifiers (variables) you will use in a script. Identifiers must be declared before they are used. FlashPoint recommends that you declare all of your variables near the beginning of the script. This makes them easier to find when reading a script. However, as long as the identifier is declared before it is used, it may be declared anywhere within the body of a script.

Declaration statements begin with the word “declare” and are then followed by an identifier type, a colon (:) and a list of identifier names.

Identifiers have the following naming rules:

- The first character of an identifier must be an upper or lower case letter of the alphabet, either A through Z or a through z.
- All characters in the identifier name after the first may be any combination of letters, numbers, and underscore (_) characters.
- Upper and lower case letters are not interpreted as the same. Because of this, Foo and foo are two different identifiers.
- Identifiers may be any length, but Digita Script only pays attention to the first 31 characters. Therefore, AReallyLongIdentifierNameThatYouAreUsing is seen by Digita Script as AReallyLongIdentifierNameThatYo.

Declaration statements tell the Digita Script interpreter what kind of information the identifier will contain.

The allowable identifier types are:

Data Type	Description
u	An unsigned integer. This is an integer in the range of 0 to 4,294,967,296 (2^{32}). Begins with 0x for hexadecimal numbers.
i	A signed integer. An integer in the range -2,147,483,648 (-2^{31}) to 2,147,483,647 (2^{31}-1). Begins with 0x for hexadecimal numbers.
f	A fixed point number. A number in the range -32768.0000 to 32767.9999 decimal.

s	A string. A sequence of characters surrounded by double quotes. The maximum length of a string is 31. An example string is "abc123". Double quotes may be used in a string by preceding them with a backslash (\) character, for example, "abc\"123". To use a backslash character within a string, precede it with an initial backslash, for example, "abc\\123".
t	Same as and interchangeable with the s data type but with a maximum string length of 255 characters. The t data type can only be used in scripts written specifically for products compatible with Digita Script version 1.5 or newer.
n	A DOS file name. A maximum of 8 characters, followed by a period (.), and a 3 character extension all enclosed within double quotes. For example, "Test_01.CSM".
b	A bitfield. A bitfield is a 32-bit unsigned integer. Each bit of the bitfield can be either true (1) or false (0). If the value is preceded by 0x, then the value is hexadecimal. If the value is preceded by 0b, then the value is binary. For example, in the number 0b1101, the first (lowest-order) bit is true, the second bit is false, and the third and fourth bits are true.

Declaration statements have the following format:

```
declare u: variable1, dataValue
```

This example declares two unsigned integer variables named variable1 and dataValue.

Assignment Statements

Assignment statements are used to give values to identifiers. An assignment statement consists of a variable name followed by an equal sign (=) followed by a value or another identifier which has already had a value assigned.

The format of an assignment statement is:

```
variable1 = 2
```

This example sets the value of variable1 equal to 2.

Note that the right side of an assignment statement can include arithmetic operations such as addition, subtraction, multiplication, and division for identifiers of type u, i, and f. For example, the statement:

```
variable1 = dataValue + 2
```

sets the value of variable1 equal to to the value of dataValue plus 2.

For a complete list of operators and the types they can operate on, see the *Digita Script Reference*.

Conditional Statements

Conditional statements allow scripts to make decisions. A conditional statement consists of the word “if” followed by a conditional expression. The following lines contain the statements to be executed if the conditional expression is true. Finally, the conditional statement ends with the word end.

For example, the statement:

```
if variable1 == 2
  variable1 = variable1 + 1
```

```
end
```

first determines whether or not the value of variable1 is equal to 2. If it is, then it adds 1 to the value of variable1. If not, then the script continues on the first line following the line containing the word end.

Other comparisons that can be used in the conditional expression include > (greater than), < (less than), and != (not equal to).

For a complete list of conditional operators and the types they can operate on, see the *Digita Script Reference*.

Goto Statements

Goto statements allow the flow of a script to jump from one point in the script to another. This is usually done as the result of a conditional statement; if a particular condition is met then jump to another part of the script. The goto statement consists of the word goto followed by a marker to which the script execution should go.

The format of a goto statement is:

```
goto markerName
```

This example would jump to the place in the script which is identified by the marker markerName.

Marker Statements

Marker statements are used with goto statements. They identify a location in a script. Marker statements consist of a marker identifier followed by a colon (:). More script statements typically follow a marker.

The format of a marker statement is:

```
Done:
  exitscript
```

This marker example could be used by a goto statement to jump to the end of a script. Note the exitscript following the marker Done: which would be the last line of the script.

Command Statements

Command statements set up and execute the Digita Script language commands.Currently, the Digita Script language has 70 command statements that give the scripter control over all aspects of the device's operation. A command statement generally includes one or more parameters enclosed within parentheses ().

An example of a command statement is:

```
Display ("Hello, world")
```

This example would display the words "Hello, world" on the camera's display.

More details on command statements, including a complete list of commands supported by Digita-based cameras, may be found in the *Digita Script Reference*.

Chapter 2 Scripting Techniques

This chapter describes the following scripting techniques:

- error checking
- capturing images
- startup scripts
- extracting bits from a bitfield
- watermark placement
- Japanese text usage in scripts

Error Checking

Digita Script provides a convenient method of error checking. Every command statement returns an error status. This status can be used to determine whether an issued command succeeded or failed.If a command succeeds, an error status of 0 is returned. If the command fails, the error status will be a number which indicates the reason for failure. Error codes are listed in the *Digita Script Reference*. The error code returned will be a signed integer. Most often, the codes returned are positive integers.

To access the returned status code, simply call the command statement as part of a variable assignment. For example, the command:

```
declare i: status
status = 0
status = StartCapture ()
```

attempts to capture an image. If the StartCapture command is successful, the value of `status` will be 0. If unsuccessful, `status` will contain an error code that may be checked for various failure conditions.

Note that the status variable, `status`, is declared as a signed integer. The name of the status variable does not matter. This example uses the name “status” simply to clarify the purpose of the variable.

It is important that the script set the value of a status variable to 0 before using it. Not all commands return 0 when they execute successfully. This is because commands which can allow user input have to temporarily suspend execution of the script while waiting for input. the script then resumes at the line following that command and does not have the opportunity to return a value if it executed successfully. The commands for which this is true include WaitForShutter(), StartCapture(), GetOption(), GetString(), and Wait(). It is good practice to always set the status variable to 0 before using it. That way, it doesn’t matter whether the successful value is returned. If an error occurs, it will be returned promptly.

It is important to check the error status of critical operations in scripts. Errors, other than syntax errors, should not cause the script to stop operation. For example, if a script tries to capture an image and is unsuccessful, the script will continue running as if the image had been captured.

The short example below demonstrates how to check for the success or failure of an image capture. The script attempts to capture an image. If it succeeds, it tells the user that it was successful. If it fails, it informs the user of the value of the error code.

```
name "Error test"
mode 0
menu "Sample scripts"
label "Error Test"

declare i: status
status = 0

SetCaptureMode (still)

status = StartCapture ()

if status == 0
  DisplayLine ("Image capture succeeded")
  Wait (2000)
end

if status != 0
  DisplayLine ("Image capture failed. Error ", status)
  Wait (2000)
end

exitscript
```

Capturing Images

Digita Script provides a mechanism for capturing images via a script. However, if the camera is not ready to capture an image and the script tries to do so, the script will continue running as if the image capture was successful. There are several methods that can be used to determine if the camera is ready to capture an image. In all cases, it is a good idea to check the error status of a StartCapture command to determine whether or not it was successful. Refer to “Error Checking” on page 7 for more details on error checking.

When a script attempts to capture images as rapidly as possible, there are two cases that need to be checked for. The first is capturing images under user control (the user presses the shutter button) and the second is capturing images automatically via script control.

When prompting the user to capture images, the quickest way to determine whether the camera is ready to capture new images is to attempt to issue a WaitForShutter() command. The camera will not successfully execute the WaitForShutter() command until it is ready to capture another image. The following code snippet checks for this:

```
declare i: status, SYSTEM_BUSY

# Set SYSTEM_BUSY equal to the error status for a system busy error.
SYSTEM_BUSY = 12

Pict1:
  # Clear the status variable before using it
  status = 0
  status = WaitForShutter ( "Picture of front of house" )
  if status == SYSTEM_BUSY
    DisplayLine ( "Processing Pictures..." )
    # Give camera time to process images before checking again
```

```
    Wait ( 5000 )
    goto Pict1
  end
  if status != 0
    # Some other error occured. Additional error checking could occur here
DisplayLine ( "Insufficient disk space.", status )
    goto Done
  end
  SetCaptureMode (still)
  StartCapture ()
```

This particular code snippet just checks for a system busy error. This error will occur if the camera is busy processing images and isn't yet ready to capture another image. If a system busy error occurs, it waits 5 seconds then attempts to issue a WaitForShutter() command again. This method is illustrated again in the CAPTURE.CSM example script which is part of the Digita Script SDK. It may be useful to check for other errors based on what is likely to occur during the execution of a particular script.

In the case where a script is automatically capturing images as rapidly as possible, another method must be used. If you use the above method, the camera will wait until the user presses the shutter button to capture the next image. This obviously won't work when the camera is supposed to capture images automatically. In this case, the best method is to check for the ipip bit of the SystemStatus bitfield to be clear. The SystemStatus bitfield is returned by the GetCameraStatus() command. This bit will be cleared when the camera has finished processing all images.

This method is slower than checking for WaitForShutter(), and so it should not be used when capturing images manually. The reason that it is slower is that most Digita-based cameras are capable of capturing more than one image at a time. If the script checks for the ipip bit to be cleared, it will not be able to capture another image until all processing of the first image is completed. If the WaitForShutter() check is used, the camera will be able to capture another image as soon as possible, even though the camera may still be processing the first image.

The following code snippet illustrates this method. The technique used here for extracting a specific bit from a bitfield is discussed elsewhere in this guide.

```
declare u: IPIP, processing
declare b: systemStatus, captureStatus, vendorStatus

# set IPIP equal to the bitmask needed to extract the ipip bit from the
# SystemStatus bitflag returned by GetCameraStatus()
IPIP = 0x10000000

Pict1:
  # Retrieve camera status information, including ipip bit.
  GetCameraStatus ( systemStatus, captureStatus, vendorStatus )
  processing = systemStatus & IPIP

  if processing == IPIP
    DisplayLine ( "Processing pictures..." )
    # Give camera time to process images before checking again
    Wait ( 5000 )
    goto Pict1
  end

SetCaptureMode (still)
StartCapture ()
```

Startup Scripts

Digita has the capability of running a script automatically when a Digita-based camera first starts up. Such a script is known as a startup script. The basic requirement of a startup script is that it be named STARTUP.CSM. The script itself is otherwise identical to other scripts.

One important issue to be aware of is that in order for a startup script to be executed properly, the camera must be already set in the operating mode specified by the script before powering on the camera. Otherwise, the script will abort when the camera sets the operating mode. This happens right before the user gains control of the camera when it is starting up.

Note that if a Kodak® DC220 or DC260 running a startup script has firmware 1.0.2 or older, the startup script will always abort. This was fixed in firmware versions 1.0.3 and later. This issue should not affect other cameras.

Extracting Bits From a Bitfield

Some information within Digita-based cameras is stored in bitfields. In particular, the data returned by the Digita Script command GetCameraStatus() is stored in bitfields. The bitfields used are all variables declared with type ‘b’. This kind of variable holds 32 pieces, or bits, of information. Each bit is equal to either 0 or 1. If a bit is equal to 1, it is said to be set. If it is equal to 0, it is not set.

When a value is returned in a bitfield, a simple calculation needs to be performed to extract a particular bit from the bitfield. This calculation is performing an AND on the bitfield with a specific number. The number varies depending on which bit needs to be extracted from the bitfield. Understanding how this works is not important if you are unfamiliar with programming techniques. What is important is understanding how to use this procedure.

Two pieces of information are required in order to use this method. The first is the number used to AND the bitfield. This number is sometimes called a bitmask. Appendix E in the *Digita Script Reference* contains a handy listing of the bitmasks needed for each position of a bitfield. The second piece of information needed is the position of the bit needed within the bitfield. This will be a number from 1 to 32. For the purposes of Digita Script, position 1 is the leftmost bit in the bitflag. This is sometimes called the Most Signifigant Bit (MSB). Position 32, or rightmost bit, is sometimes called the Least Signifigant Bit (LSB).

Once this information is obtained, simply AND them using the & operator. For example, use the following example script to determine whether the first bit of the bitfield ‘testField’ is set. When run "The bit is not set!" will be displayed.

```
# This script appears as the item "Bitmask"
# in the "Sample Scripts" menu.

name  "bit set test"
mode  0
menu  "Sample Scripts"
label "Bitmask"

declare b: testField
declare u: bitmask, result

# Set testField to some arbitrary number for this example
testField = 1

# Set bitmask equal to the appropriate number from table in
# Appendix E of the Script Reference
bitmask = 0x10000000
```

```
# Perform the AND of the two numbers and check to see if the bit is set
result = testField & bitmask

if result == bitmask
    DisplayLine ( "The bit is set!" )
    Wait ( 3000 )
end

if result == 0
    DisplayLine ( "The bit is not set!" )
    Wait ( 3000 )
end
exitscript
```

Watermark Placement

Placing watermarks on an image is a useful feature of Digita-based cameras. The method used to determine where a watermark is placed on an image requires a little bit of explanation.

When specifying where a particular watermark element should be placed on an image, a number from 0 to 100 is used for both the x and y positions. This number is the percentage of total distance from a specific corner. The x axis increases to the right and the y axis increases to the left. So, the upper left of the image is 0,0 and the lower right is 100,100.

When specifying the position of a watermark element, the following rules apply:

- for x or y percentages less than 50%, the number represents the percentage distance from the top left
- for x or y percentages greater than 50%, the number represents the percentage distance from the bottom right
- for x or y percentages equal to 50%, the element is centered on that axis.

The reason for this is that when starting at 0,0 the left/top corner of the watermark element is positioned in the upper left of the image. If the position increased in this way past the mid-point of a particular axis, the element might run off the edge of the screen on the right or bottom edge. To prevent this, when the percentage passes the mid-point, the camera starts calculating distance from the right/bottom corner of the watermark element. So, placing a text watermark at position 51,51 would put the the right/bottom corner of the text just below and to the right of the center of the image.

This following example script places the text "Welcome to FlashPoint!" at the top-center of an image.

```
# This script appears as the item "Watermark"
# in the "Sample Scripts" menu.

name  "Setting the default location for watermarks."
mode  0
menu  "Sample Scripts"
label "Watermark"

declare u: wena, status

status = 0

# Set the bit mask to the value required to turn the watermark text on.
wena = 0x40000000

# Set the error checking variable to the return value from SetCameraState.
```

```
status = SetCameraState ( "wena", wena )

# If an error occurs, wait 3 seconds then exit.
if status != 0
    DisplayLine ( "Error enabling watermarks: ", status )
    Wait ( 3000 )
    goto Done
end

# Set the horizontal and vertical position of the string watermark
# to (50,0).
# Set the watermark string to "Welcome to FlashPoint!"

status = SetCameraState ( "wsxp", 50 )
status = SetCameraState ( "wsyp", 0 )
status = SetCameraState ( "wstr", "Welcome to FlashPoint!" )
status = SetCameraState ( "wsop", 1 )

# Inform camera user that the "Welcome to FlashPoint!"watermark is
# is enablec, then wait 3 seconds to continue.
DisplayLine ( "FlashPoint watermark enabled." )
Wait ( 3000 )

Done:
exitscript
```

Japanese Text Usage In Scripts

The following method can be used to display Japanese characters from a script. This will only work on cameras which contain Japanese support. U.S. versions of some cameras do not contain such support.

Some important points to be aware of in the use of Japanese text with Digita Script are:

- Japanese characters must be entered as hexadecimal values within a string using the format \x11, where \x tells Digita Script that an extended character is being used and 11 is any valid hexadecimal number between 1 and 255. A table showing the mapping of these numbers to characters in the standard camera font can be found in Appendix F of the *Digita Script Reference*.
- Japanese text cannot be placed in the script menu or label.
- Japanese text may only be used with the Display, DisplayLine, SetOption, and Alert commands.
- If the two bytes following the \x cannot be converted to a numeric value (e.g. "\xfp") a syntax error will be displayed.
- In order for Japanese text to be displayed, the region code parameter (rgnc) must be set to 8.
- Japanese and English text can be mixed in the same string.

The following script is an example of how to use Japanese text in a script. If the camera the script is loaded on supports Japanese characters, "Hello world" will be displayed in Japanese characters.

```
# This script appears as the item "Japanese Text"
# on the menu "Example Scripts"

name "Japanese Text Test"
mode  0
menu  "Example Scripts"
label "Japanese Text"

declare u: status, Ro_rgnc
```

```
# Capture original region code
GetCameraState ( "rgnc", Ro_rgnc)

# Set to the Japanese region code
# and set up an error checking variable.
status = SetCameraState ( "rgnc", 8 )

# Camera suports Japanese characters.
if status == 0
   DisplayLine ( "\x1A\x9A\x8A\xC3\xBD\xC4\x83\xDE\x1D\xA1" )
   Wait ( 6000 )

# Return camera to its original state
   SetCameraState ( "rgnc", Ro_rgnc)
   DisplayLine ( "Camera reset to prescript region code." )
   Wait ( 3000 )
end

# Camera does not support Japanese characters.
if status != 0
   DisplayLine (" Japanese  characters not supported.")
   Wait ( 6000 )
end

exitscript
```