FotoScript: The Language Reference Manual

Matthew Raibert
mjr2101@columbia.edu

Norman Yung
ny2009@columbia.edu

James Kenneth Mooney
jkm2017@columbia.edu

Randall Q Li
rql1@columbia.edu

October 23, 2004

Here is a description of the various features of FotoScript and how to use them. This manual is based in part on The C Programming Language by Brian W. Kernighan and Dennis M. Ritchie:

1 Tokens

There are five classes of tokens in FotoScript: identifiers, keywords, literals, separators and numerical operators. Blanks, tabs, and comments are ignored except as they separate tokens.

1.1 Comments

As an experiment, we have chosen to implement comments in an unorthodox way. All capital letters will be ignored by FotoScript with the exception of those within string literals. A user will be able to write comments in all caps anywhere in the code. Because this is unusual, and possibly really annoying, it is subject to change.

1.2 Identifiers

Identifiers are made up letters and digits. The first character must be a letter; upper-case characters are ignored by fotoscript and cannot, therefore, be used in identifiers.

1.3 Keywords

Fotoscript reserves several keywords:

```
break data depth format else foreach foto fotos height if modified
new open set while name width
```

1.4 Literals

There are two kinds of literals: string literals and number literals.

1.4.1 string literals

A string literal is a string of characters surrounded by double quotes.

1.4.2 number literals

A number literal is a string of digits which represent a non-negative decimal integer. A user may use operators to create negative numbers as well as fractional numbers.

1.5 Separators

Separators include curly braces and newline characters. The curly braces, '{' and '}', designate blocks of code. The newline character separates expressions.

2 Types

In Fotoscript there are three classes of objects: images, numbers and strings. Variables are implicitly typed when they are created. For this reason, when it is created, each variable must either be assigned the value of a literal or that of an already typed variable; it takes it's type from whence it's initial value is taken.

2.1 The Image Class

Fotoscript has a native class of objects for storing all the data related to an image. This object has the following fields:

Field Name	Brief Description
`data`	the raw image data
`name`	the file name
`width`	the image width in pixels
`height`	the image height in pixels
`depth`	the image color depth in bits per pixel
`format`	the file format for the image to be saved to disk
`modified`	set to one if the image has been modified, zero otherwise

2.1.1 foto

For convenience, FotoScript has a special pointer to an image object; it signifies the current image. It can be assigned to point to a certain image by the user and is assigned automatically by some funtions. It is used by foreach to allow the user to iterate through all the images.

2.1.2 fotos

Because it may be useful to load and edit several images at once, there is a built-in array of image objects called `fotos`. When a new image object is loaded or created, it is appended to `fotos`. The images in `fotos` can then be accessed by `foreach` as described below.

2.2 The Number Class

FotoScript does not distinguish between different classes of number. A number variable can contain a reasonable range of positive and negative floating point numbers. Only non-negative integer values can be represented by string literals, but there are some operators that cause number literals to behave as one might expect. Here is a table of number operators; those with highest precedence are higher; each precedence level groups left to right:

Operators
unary and binary `.`
unary `-` and `+`
`*` and `/`
binary `-` and `+`
`==, !=, <, <=, >, >=`

2.2.1 The Unary - Operator

The unary - operator negates the number that follows it. Because their functions are similar, it does not need to be strongly distinguished from the binary - operator; if there is only one sensible operand, the unary version is used; if there are two, the binary version is used; no sensible operands is an error. For example: the string `-25` returns a negative number with magnitude twenty-five.

2.2.2 The Binary - Operator

The binary - operator has the same function as the unary version except that after the number following the operator is negated, it is added to to the number preceeding the operator. For example: the string `10-25` returns a negative number with the magnitude fifteen.

2.2.3 The Unary . Operator

The unary . operator divides the number that follows it by 2^l where l is the length of the shortest string representing the number in decimal form. Because their functions are similar, it does not need to be strongly distinguished from the binary . operator; if there is only one sensible operand, the unary version is used; if there are two, the binary version is used; no sensible operands is an error. For example: the string `.25` returns a number with the value twenty-five hundreths.

2.2.4 The Binary . Operator

The binary . operator has the same function as the unary version except that after the number following the operator is divided as described above, it is added to to the number preceeding the operator. For example: the string `10.25` returns a number with the value ten and twenty-five hundreths.

2.2.5 The Binary + Operator

The binary + operator adds its two operands together. If there is only one sensible operand it will have no function; no sensible operands is an error. For example: the string `10+25` returns a number with the value thirty-five.

2.2.6 The Binary * Operator

The binary * operator multiplies its two operands together. Fewer than two sensible operands is an error. For example: the string `10*25` returns a number with the value two-hundred-and-fifty.

2.2.7 The Binary / Operator

The binary / operator divides the operand preceeding it by the operand that follows it. Fewer than two sensible operands is an error. For example: the string `10/25` returns a number with the value ten twenty-fifths.

2.2.8 The == Operator

The == operator compares its two operands and returns zero if they are equal and one otherwise. Fewer than two sensible operands is an error. For example: the string `10==25` returns one.

2.2.9 the != Operator

The != operator compares its two operands and returns one if they are equal and zero otherwise. Fewer than two sensible operands is an error. For example: the string `10!=25` returns zero.

2.2.10 The <Operator

The <operator compares its two operands and returns zero if the latter is greater than the former and one otherwise. Fewer than two sensible operands is an error. For example: the string `10<25` returns zero.

2.2.11 The <= Operator

The <= operator compares its two operands and returns zero if the latter is greater than or equal to the former and one otherwise. Fewer than two sensible operands is an error. For example: the string `10<=25` returns zero.

2.2.12 The >Operator

The >operator compares its two operands and returns zero if the latter is less than the former and one otherwise. Fewer than two sensible operands is an error. For example: the string `10>25` returns one.

2.2.13 The >= Operator

The >= operator compares its two operands and returns zero if the latter is less than or equal to the former and one otherwise. Fewer than two sensible operands is an error. For example: the string `10>=25` returns one.

3 Syntax

Fotoscript syntax is very simple. Each line of code can be of one of two types: flow control and functions.

3.1 Function Syntax

Function syntax is simple. The function name is written followed by some number of identifiers and it is terminated by a newline character. Some examples of functions are the following keywords:

3.1.1

`set` $variable$ $value$
sets $variable$ to $value$; variables have three types: number, string and image; the literals for each type are non-ambiguous so the variable types are implicitly defined; $value$ can either be a literal or a variable that has already been implicitly typed by setting it with a literal.

3.1.2 break

`break`
break out of the current control loop.

3.1.3 open

`open` $name$
opens a file or directory called $name$; if $name$ is not a readable file or directory it is an error.

3.1.4 new

`new` $name$
creates a new image for editing called $name$; note that it does not write the image to disk unless `save` is called.

3.1.5 save

`save` $image$
writes $image$ out to disk.

3.2 Flow Control

Special keywords control the flow of the program. Here are descriptions of each of these keywords and their syntax. $test$ can be omitted and will, by default, evaluate to true.

3.2.1 foreach

`foreach` $test$ $\{statements_1\}$ `else` $\{statements_2\}$
For each member of `fotos`, set `foto` to that member and perform $statements_1$ if $test$ is true, $statements_2$ otherwise.

3.2.2 if

`if` $test$ $\{statements_1\}$ `else` $\{statements_2\}$
Perform $statements_1$ if $test$ is true, $statements_2$ otherwise

3.2.3 while

`while` $test$ $\{statements\}$
Repeat $statements$ as long as $test$ is true.