# uBASIC User Guide

## (from 'UBASIC/TutorialScratchpad' @ CHDK Wiki)

Draft 0.1, 15 June 2009, DanielF

# Contents

# Starting Out

**Keep these things in mind when writing your scripts:**

- Use any text editor to compose your scripts. Make sure it is saving the file as TEXT-ONLY. Do NOT use programs like Word or other advanced editors. These insert hidden header info and use non-standard ASCII characters for Return/Line-Feed commands, quotation marks, and others. The simplest of text-editors will suffice, even then watch out not to use TAB for indenting. (Notepad in Windows, nano in Linux for example.) Mac users, make sure your script is in UTF-8 encoding, see this special note concerning Macs and Script Files.
- Keep all commands in lower-case. Variables are case-sensitive (a and A are not the same).
- In versions prior to 0.5.5 (SVN Changeset 524) you are limited to 8k (8192 bytes) for the size of your script – CHDK Build 119 or later. (2k (2048 bytes) in CHDK Build 116 or earlier.)
- Be aware that not all commands work on all cameras, if you plan on sharing your script try to keep it as generic as possible unless you have a special need for the camera-specific commands. Try to also provide a more generic version so that all may benefit from it.
- If using earlier CHDK Builds some commands listed here will not be available to you and cause errors, this tutorial will be updated as new commands and changes are made to CHDK.
- **Keep your script concise and efficient!** It takes 10ms of time for each line of your script to be interpreted by tiny uBASIC. If your script is 10 lines long this takes 1/10th of a second, 100 lines takes a full second, etc. This can greatly impact high-speed uses. Even **rem** statements take 10ms to be processed, use them sparingly. See this section in the discussion area for **script timing test results** for further info. In versions 0.5.5 and later, up to 100 rem statements and labels will be executed before a 10ms wait is required.
- If you write an interesting script, **please** share it with the rest of us on the User Written Scripts pages so we may learn from you! Beginner script-writers can be some of the most creative!
- See these pages for some ideas and examples if you are just starting out: **User Written Scripts**
- Two new Scripts Menu options have been added to some special builds, read about them in the Special Builds Features on the firmware usage page. By using these two options in conjunctions with these scripts, you are able to execute any script when first powering on your camera. This allows you an unlimited number of favourite Custom shooting modes and USB-Remote functionality. You may want to write your scripts taking these extra features into account.

# The Script Header

When viewing scripts you'll often see the opening section look something like this:

```
@title Interval shooting
@param a Shoot count
@default a 5
@param b Interval (Minutes)
@default b 0
```

Let's break down what each of those lines mean and how they are used by CHDK.

```
@title Your Script Title
```

This is the title that will appear when you have the script loaded in CHDK and go to "Scripting Parameters" menu selection. It will appear under the line "----Current Script----" as well as in the lower-left corner of your viewfinder when you are in <ALT> mode. Keep your title short (24 characters or less). Otherwise the title will cover up the <ALT> label.

```
@param x (label)
@default x n
```

This is where you will define the beginning values of any variables used in your script. These are often used to set how many exposures you want, how long of a delay you want, how many bracketing steps you want, etc. These variables can be changed by the end-user from the "Scripting Parameters" menu items. In that sub-menu, they will appear under the line "----Script Parameters-----"

```
@param x (label)
```

The "**x**" in that line will be one of any lower-case Latin letter from **a** to **j**. The (label) is the text string that will appear in your "----Script Parameters----" list, to let the end user know which of the variables they are changing (i.e. number of shots, how many steps, etc.).

Up to 10 @param statements, user-controllable variables, may be used in any one script.

*Note: The latest builds of CHDK now allow you to have up to 52 variables, **a** to **z** and **A** to **Z**. But the user-definable variables must be in lower-case if used for that purpose. Also be aware that lower and uppercase variables are unrelated. If you use a lower-case **j** for a variable, it is not the same as using **J**, and vice-versa.*

```
@default x n
```

This statement sets up the default, or beginning value of your (lower-case letter) variable, where "**x**" is the variable from the @param statement above, and "**n**" is the default value to start with. This value is only used when a script is loaded for the first time.

**Notes:**

If there is no **@title** command the filename of script is used. If there are no **@param** commands CHDK assumes that there are three adjustable variables: **a**, **b** and **c**. Remember – when naming **@param** variables, use only a character from **a** thru **z**.

After your default variable values have been defined here, it is good to add some lines right after this section to ensure those numbers will be used in case the user has input 0 (zero) for some value that needs to be higher (or lower). You will see this in scripts as:

```
if a<2 then let a=5
```

If your default value that you wanted the user to start out at for parameter variable **a** was 5, then if they left that setting at 0 (zero) or 1, then this will automatically increase that variable's value back up to 5.

After you have set up your variable parameters, then comes the crux of your script, the part that does the actual work and tells the camera what to do, when to do it, and what buttons or commands need to be executed. Since we are working with a very small subset of the larger uBASIC programming language, it might be good to list and explain only those that are available to the CHDK script writer.

# The Basics of BASIC Programming

## *Logic Commands*

All programs are designed to mindlessly repeat some commands. In order to make them work in the proper order, and the correct number of sequences, they have to be contained in some simple recurring loops and counters. Testing for when some condition has been met, before it can go onto the next command, or finally end the program (script).

There are several ways this can be done in BASIC, by using numeric counters, and loops. There are some built-in commands to simplify these tasks.

### The LET Command

This one is simple. If you see a command that says "let **a** = 2" then that's exactly what happens. It defines the value of 2 to the variable **a**.

This command is mostly included just for legibility. You can leave off the **let** command and it will still work. Example: **let a=2** can be written more simply as **a=2**. Or this example: **if z>5 then let b=0** can be simplified to **if z>5 then b=0**. Doing so will greatly save on script space if you have to define and redefine many variables many times.

### The IF / THEN / ELSE Commands

These are used to test for the truth of a certain condition. **IF** something is true, **THEN** this takes place, **ELSE** (otherwise) do this if it is not true.

A simple example:

```
if a > 2 then goto "subroutine1"
```

If in your script, the variable **a** has been assigned to a value greater-than 2, then the script will jump to the labelled subroutine1.

```
if a >2 then goto "subroutine1" else goto "subroutine2"
```

In this case if **a** is NOT greater than the value of 2, your program will jump to subroutine2.

The conditional expressions allowed in uBASIC are: = (equal to), > (greater than), < (less than), <> (not equal to), <= (less than or equal to), >= (greater than or equal to)

### The FOR / TO / NEXT Commands

These are used to set up simple loops. You will often see them in scripts as in this example:

```
for n=2 to a
    sleep t
    print "Shoot", n, "of", a
    shoot
next n
```

The first line "for **n**=2 to **a**" means that the "for / to / next" loop will run while variable-**n** equals the sequence of numbers of 2 up to whatever the number variable-**a** has been assigned to. The commands that take place in the loop are contained between the FOR statement and the NEXT statement. "**next n**" tells the loop to go back to the beginning "**for ...**" statement until the **a** value has been reached.

For example:

```
for n=1 to 10
   print "This is line number", n
next n
```

This will produce the sequence of:

```
This is line number 1
This is line number 2
This is line number 3
.
.
.
This is line number 9
This is line number 10
```

and then that loop will end and go on to the next sequence of commands.

## Subroutines using GOSUB (and related GOTO) Commands and Labels

### Sub-Routines

For complex programming tasks, it is often helpful to split the program into smaller subroutines that can be called with **gosub** and **goto** commands. A sub-routine can be nearly anything but it is generally used for a set of commands that will be called-up more than once. Instead of writing the same set of commands over and over again you put that code into a **subroutine** and then call it up from within the main program by using **gosub "label"** or **goto "label"**. Subroutines are generally placed after the main code. A labelled **subroutine** that will be called by **gosub "label"** needs to end with the **return** command, to tell the script to jump out of that section of code and return back to from where it was called.

**GOSUB** and **GOTO** are similar but you should refrain from using **GOTO** unless you know what you are doing. **GOSUB** will always return from a subroutine as soon as it reaches the **RETURN** command. **GOTO** does not behave this way. **GOTO** should only be used when you are going to jump to a section of the script one time and under special circumstances.

### GOSUB and GOTO Examples

A simple **GOSUB** example (the subroutine's label and subroutine are in bold):

```
for x=1 to 10
  gosub "display"
next x

:display
  print x
  return
```

A longer example that would capture 3 images with increased ISO settings would look something like this:

```
shoot
for i=1 to 3
  gosub "incISO"
  shoot
next i
for i=1 to 3
  gosub "decISO"
next i
end
```

```
:incISO
  click "menu"
  [some more clicks]
  return

:decISO
  click "menu"
  [some more clicks]
  return
```

An example using the **GOTO** command taken from an endless intervalometer script. NOTE: This situation creates an endless loop. Until you manually override the script it will continue. This is generally considered BAD FORM! Any script should include/end-with all the commands to reset the camera to its original configuration prior to running the script, and properly end with the **END** command. Do not do this kind of thing unless you have a special need for it and know what you are doing.

```
@title Interval Shooting Non-stop
@param a Interval (Minutes)
@default a 0
@param b Interval (Seconds)
@default b 5
@param c Interval (10th Seconds)
@default c 0

t=a*60000+b*1000+c*100

if t<100 then let t=5000

n=1

print "Interval shooting."
print "Until you interrupt it."
print "Use with caution."

sleep 1000

:shot
  print "Shot number", n
  shoot
  n=n+1
  sleep t
  goto "shot"
```

## The "print" Command

This will print whatever text follows the statement to your LCD or EVF display in the mini-console area (see firmware usage) while the script is running.

Syntax: **print "25 characters of text"**

**Note:** You are limited to 25 characters being displayed (without wrapping) in any one line of text. You may also include the values of variables or integer-equations in your **print** statement. CHR$() is **not** supported (indeed, it crashes CHDK!), nor is PRINT USING…

Examples:

```
rem Print total duration of interval to viewfinder
print "Total time:", t*a/60000; "min", t*a%60000/1000; "sec"
sleep 1000
rem Start actual camera operation in a loop
print "Shoot 1 of", a
shoot
for n=2 to a
    sleep t
    print "Shoot", n, "of", a
    shoot
next n
```

Note that the comma (,) is replaced in the output with a space while a semicolon (;) results in no whitespace.

Example:

```
print "C","H","D","K"
print "C";"H";"D";"K"
```

will result in

```
C H D K
CHDK
```

### The "print_screen" Command

Whatever the script prints on the mini-console screen is also written to file '/CHDK/SCRIPTS/PR_SCREEN.TXT'.

First call is either,

**print_screen 0** The text is appended to the last file. If the file was there already, the text is written at the end and the older text is not removed.

**print_screen 1** The text is written to "A/CHDK/BOOKS/PS00000.TXT". The new text overwrites any existing text in the file if there was any .

**print_screen N** The text is written to the next file number. The file number cycles between 0 and N-1. If the resulting file number is 5, then the text is written to file "A/CHDK/BOOKS/PS00005.TXT". The file number of the last written file is kept in file "A/CHDK/BOOKS/PS_COUNT.TXT". Delete the file to reset the counter.

*(The print_screen N statement is not implemented in the Allbest builds !)*

**print_screen 0** turns off writing to the file and **print_screen 1** turns it back on.

Example:

```
@title printscreen test

@param a None
@default a 0

@param c mode: 0-append, 1-replace, other-modulo c
@default c 1

print_screen c
print "START "c
print_screen 0
print "Not written to file"
print_screen 1
print "This should be written to the file."
print "a="a
print_screen 0
end
```

### The "cls" Command

CLS stands for "Clear Screen". This is used to clear the mini-console screen from any "print" statements in an easy way. Instead of having to issue 5 command lines of **print " "**, you just need to issue this one small **cls** command.

### The "Sleep" Command

This pauses the script to allow some action to take place, or to delay when the next action should occur.

Syntax: **sleep x**

Where **x** is any variable or whole number. The value is in 1000ths of a second, but timer resolution is only around 10–30 ms.

Example: **sleep 1500** means to pause for 1.5 seconds.

### The "REM" Command

The "rem" (which stands for "remark") command is sometimes used to place comments in a script. It is only used as a reminder for the person writing or viewing the script. Like an internal note. This command is not executed nor seen when the script is run. However, keep in mind that scripts for CHDK can be only 8k (8,192 characters) in length. (Only 2k in CHKD before Build 119.) Too many REM statements can slow down your script as well as taking up valuable space.

An (overzealous) example of REM commands in a script:

```
rem Interval shooting

@title Interval shooting
@param a Shoot count
@default a 10
@param b Interval (Minutes)
@default b 0
@param c Interval (Seconds)
@default c 10

rem Calculate 1000ths of seconds from variables:
t=b*60000+c*1000

rem Sets some default variables to initial values:
if a<2 then let a=10
if t<1000 then let t=1000

rem Print total duration of session in viewfinder:
print "Total time:", t*a/60000; "min", t*a%60000/1000; "sec"

rem Delay actual shooting so they can read the above print statement:
sleep 1000

rem Start actual camera operation in a loop:
print "Shoot 1 of", a
shoot
for n=2 to a
    sleep t
    print "Shoot", n, "of", a
    rem This takes the actual exposure:
    shoot
next n

rem Ends this script

end
```

**REM** statements can always be removed from a script if you feel there are too many or unneeded. Removing a **rem** line will not impact the operation of the script in any way (other than speeding it up and using up less memory space).

### The "exit_alt" Command

This command leaves the <Alt> mode.

### The "End" Command

This should be the last line in your script. It tells the script to cease all operations and return camera control back to you. Before **end**ing a script, it is good form to always reset any camera settings that the script took control of during initialization or running of your routine, so that the end user doesn't have to undo all the key-presses and menu changes that the script created.

## *Special Build Commands*

Due to the open-source sharing of this project, many other talented individuals have been creating their own versions of CHDK, some with exceptional improvements or features that don't exist in the original CHDK. An attempt will be made to include the commands of those builds that have important features worth considering. Please note that any commands that appear in the "Special Builds" sections in this tutorial will not work with the original CHDK by GrAnde, unless he sees fit to include them in his own builds one day.

### Fingalo's Builds

Available from: Fingalo's CHDK2

#### FOR / TO / STEP / NEXT Loops

A standard BASIC **step** command was added to the **for/to/next** commands make loops easier. Instead of using multiple lines for counters to increment numeric expressions with commands like a=a+1 or b=b-3, a simple **next** command may now be used.

Usage:

```
for var=expr to expr step expr
statement
statement
statement
...
next var
```

Where **var** can be any variable, **expr** can be any defined variable or math expression, and **step** can be any defined variable or math expression. The step value may also be negative to increment in reverse.

Example:

```
@title Focus Bracket Steps
@param d Near Focus (mm)
@default d 2500
@param e Far Focus (mm)
@default e 4500
@param f Step Increment (mm)
@default f 100

for x=d to e step f
  set_focus x
  shoot
next x

end
```

If using the default values this simple script will start out at the Near Focus value of 2500mm, increment that value by 100mm every time, shoot an image, and exit when the focus has reached or passed 4500mm.

### Do / Until Loops

Another method of creating loops for repetitive instructions or when waiting for some condition to be true. Code within a Do/Until loop will always be executed at least once (unlike While/Wend loops)

Usage:

```
do
statement
statement
statement
...
until relation
```

Where **relation** may be any logical expression. When it is true, the loop will exit.

Example:

```
rem set some starting values for the variables
y=0
x=5

rem start do-loop

do

rem increment x by 10 each time
x=x+10

rem increment y by 1 each time
y=y+1

rem print results to viewfinder mini-console
print "This DO loop happened", y; "times."

rem repeating do-loop until x is equal to the value of 55
until x=55

end
```

### While / Wend Loops

Similar to the DO / UNTIL loops. The loop will continue to execute **while** some statement remains true, and will end, **wend** (while-end), when that statement is no longer true. Unlike Do/Until loops, code within a While/Wend loop may never be run, if the test condition is already false when the While statement is first reached.

Usage:

```
while relation
statement
statement
statement
...
wend
```

Example:

```
x=0
while x<200
  x=x+25
  print "The value of x is", x
wend
```

This loop will increment the value of **x** by 25 each time and print the value of **x**, as long as (while) the variable **x** remains less-than 200

### IF / THEN / ELSE / ENDIF – Multiple Statements

Fingalo reports: "Seems to have some bug when not using the else in nested if constructs!"

Usage:

```
if relation then
statement
statement
statement
...
else
statement
statement
statement
...
endif
```

The standard single-statement **if**...**then**...**else**... loop still works, but it cannot be used inside the **if**...**then**...**else**...**endif** loops.

NOTE: nesting levels for all loop methods are currently set to 4 for all new constructs.

### IS_KEY Optional Method

Also added a variation of the **is_key** statement, so **is_key** can be used as:

```
if is_key "set" then goto "continue"
```

And also as:

```
k = is_key "set"
```

The original statement version (example below) may still be used.

```
is_key k "set"
if k=1 then goto "continue"
```

The main reason for this new '**is_key'** option and other loop methods is that you can now more easily make key-press detection loops. Such as:

```
do
  if is_key "right" then gosub "r_label"
  if is_key "left" then gosub "l_label"
until is_key "set"

rem begin r_label subroutine
:r_label
  (commands)
rem begin l_label subroutine
:l_label
  (commands)
```

### Microfunguy's SDM (Stereo Data Maker) Builds

Microfunguy has reduced the number of 'standard' uBASIC programming commands to those more commonly used.

He has also added a number of 'plain English' commands that simplify continuous and custom-timer bracketing sequences.

The example below uses the number of images set in customer-timer menu and takes a bracketed sequence such that each image is 1 stop darker than the previous one:

```
hdr_bracket_1/3_ev_steps 3
each_photo_darker
" Press switch"
wait_for_switch_press
" Each image darker"
start_custom_timer_sequence
wait_until_done
```

## uBASIC variables

Variables are represented by single letters of the Latin alphabet: a–z (in some versions, A–Z also). All variables are 32-bit signed integers (–2147483648 to +2147483647).

## Labels

A label must be the only statement in a line and start with a colon(**:**).

## Math Expressions allowed in uBASIC

Build 144:

| | |
|---|---|
| + | Addition |
| – | Subtraction |
| * | Multiplication |
| / | Division |
| % | Remainder (explanation see below) |
| < | Less Than |
| > | Greater Than |
| = | Equal |
| <= | Less Than or Equal (CHDK Build #144 or later) |
| >= | Greater Than or Equal (CHDK Build #144 or later) |
| <> | Not Equal (CHDK Build #144 or later) |
| & | And |
| \| | Or |
| ^ | Xor |

Remember CHDK before build 144 could only use: +, **-**, *, /, **%**, |, **&**, <, > and = operations, and just simple assignment (**=**).

Most of the expressions are easy to understand, but the **%** (remainder) operation might need a short explanation.

> Example: Let's say you have computed a number to equal how many seconds something will take (its duration). Such as **s=**(some math expression) Where **s** is being assigned the number of seconds computed.

Now you want to display that as minutes and seconds. You will need a print statement such as:

```
print "Total Time:" , s/60; "min", (the remainder of s/60); "sec"
```

There is a very simple way to do this using the % command. Think of % as "the remainder of s being divided by". So all you need to do is have this print statement:

```
print "Total Time:" , s/60; "min", s%60; "sec"
```

If **s** had the value of 328 seconds, then this would print out to:

```
Total Time: (328/60)=5 min (the remainder of 328/60)=28 sec
```

or more simply

```
Total Time: 5 min 28 sec
```

**Some further notes:**

< Less Than
> Greater Than
= Equal
<= Less Than or Equal
>= Greater Than or Equal
<> Not Equal

are relational operators, while

& And
| Or
^ Xor

are bitwise operators, not logic operators. (The logic operators of **and**, **or**, and **not** have been added to CHDK build #144 or later.) Example use of the bitwise &, |, and ^ binary operators are:

```
e=5|3
print e
```

will return "7".

`5&3` will result in "1"

`5^3` will result in "6"

For an explanation refer to bitwise operators

## *Logical Operators: AND, OR, NOT*

not logical **not** (best to use in form with parentheses ie. not (expression),

and logical **and**

or logical **or**

Priority for evaluation order has been updated so expressions like

```
if a=1 or b=3 and c>4 or d<>7 then ...
```

are being correctly calculated, although one would preferably use parentheses just to understand what is being calculated.

Also priority for "**&**" and "|" has been changed the same way.

NOTE: Multiple relational operators are allowed!

## Camera Operation Commands

These commands are designed to allow your script to control your camera much like you would manually. Nearly anything you can do by pressing buttons on your camera with your own fingers, you can also do automatically with these script commands. The complexity and time-line of your script is only limited by your imagination and trying to keep your script under the 8K character (8192 byte) limit.

Camera commands can be written in 3 flavours / command-methods:

- **click "button-name"**

  Presses the button momentarily, used for one time, instantaneous commands. This will be the most often used method of issuing a camera command.

- **press "button-name"**

  Presses and **HOLDS** the required camera button, it remains pressed until the same button-name is given the **release** command. Some camera commands can only be accessed when a button is held down during use.

  Example: In Manual Focus in the S-series cameras the **MF** button needs to be held down while the focus commands are being issued. Or when shooting in high-speed burst mode, then the shutter button must be held down during its needed duration with the **press "shoot_full"** command.

- **release "button-name"**

  Ends the **press "button-name"** command. If using a **press "button-name"** command be sure to end it with the **release "SAME-button-name** command at the appropriate sequence in your script to reset things back to normal.

All camera command buttons that you can press manually you may use in your scripts using this syntax. The **only** exception is the often-used **shoot** command. **shoot** is used by itself without the leading **click**, **press**, and **release** command methods.

All button-pressing commands (except **shoot**) should be written in the following syntax:

```
command-method "button-name"
```

where command-method may be **click**, **press**, or **release**, and the button-name must be enclosed in double-quotes.

For example, a simple script using all 3 command-methods which makes AELock and AFLock on A-series cameras:

```
sleep 2000
press "shoot_half"
sleep 1000
click "erase"
click "down"
release "shoot_half"
```

### *shoot*

Records an image.

This command is similar to the **click "shoot_full"** command (see below), but it waits for the camera to perform some normally automatic actions, such as auto-focusing, charging the flash, etc. For example: if in AUTO, P, Tv, Av, or any SCN modes, using the "shoot" command causes the camera to check focus and exposure for each shot. When "shoot" is used in intervalometer scripts this far surpasses the camera's own built-in intervalometer in that the camera only sets exposure and focus once for the initial exposure, as if it was only using the "click 'shoot_full'" command. This "shoot" command in an intervalometer script allows it to compensate for all the things that can change over the course of many minutes and hours. For more precise manual control of the camera in scripts, see the **click "shoot_half"**, **click "shoot_full"**, when used in conjunction with the **get_tv, set_tv, set_tv_rel, get_av, set_av, set_av_rel** commands below.

## *click/press/release "up", "down", "left", "right"*

Actuates the respective directional button of your "Omni-Selector" (navigational buttons).

## *click/press/release "set"*

Actuates the set button.

Note: **press** and **release** would not normally be used with this button, but without knowing each and every camera model's functions and the creative ways some might use scripts, these two command-methods are also mentioned.

## *click/press/release "shoot_half"*

Actuates the shutter-release in the half-press position. This is often used to lock focus, exposure, or other camera settings.

(Note: In dim light it can sometimes take up to 2+ seconds for a camera to hunt for focus. If your script is using this command to set auto-focus, and is designed for or intended to also be used in low-light conditions, it would be good to follow a **press "shoot_half"** command with a **sleep x** command, where x can have a value from 1500 to 2500.)

## *click/press/release "shoot_full"*

Actuates the shutter-release button completely, regardless of whether the camera has finished charging the flash or other normally automatic camera operations.

## *click/press/release "zoom_in", "zoom_out"*

Initiates your camera's zoom control one zoom-step at a time. (It is uncertain at this time (I didn't test it), how this will act using the **press** and **release** commands.) The A-Series cameras have 9 or 15 zoom steps (0 to 8/14), and the S-series cameras have 129 zoom steps (0 to 128). This command may require an extra sleep command after each zoom step. When using **click** the S-series cameras implement this command very slowly. Here's an example of how it may be used in a loop:

```
for s=2 to a
  for n=1 to b
    print "Zooming-in ", n; "..."
    click "zoom_in"
    sleep 600
  next n
  print "Shoot", s, "of", a
  shoot
next s
```

Note the 0.6 second **sleep** command after each **zoom_in** step.

## *click/press/release "menu"*

Actuates the menu button.

This is used to alter some of the cameras settings that can only be set through the record menus, to set up the camera before a script-session, or during.

Note: **press** and **release** would not normally be used with this button, but without knowing each and every camera model's functions and the creative ways some might use scripts, these two command-methods are also mentioned.

Example:

```
:slowsync
  click "menu"
  sleep 400
  click "down"
  sleep 400
  click "down"
  sleep 400
  click "down"
  sleep 400
  click "right"
  sleep 400
  click "menu"
  sleep 400
return
```

This **:slowsync"** sub-routine will initialize the camera's flash setting into slow-sync mode. Note also the **sleep** commands, giving your camera time to respond to the new settings between each directional button-press. Button-press delay times may be camera specific. (Meaning it might be a good idea to set up a user-defined variable for these in some scripts to save on script-size and make the script more adaptable to more makes and models of cameras. A note could be made in the accompanying script's documentation on what button-press delays are needed per make and model of camera.)

## *click/press/release "display"*

Actuates the camera's display button.

Note: **press** and **release** would not normally be used with this button, but without knowing each and every camera model's functions and the creative ways some might use scripts, these two command-methods are also mentioned.

## *click/press/release "print"*

Actuates the camera's print button. (Note: actuates the shortcut button for S-series cameras.)

Note: **press** and **release** would not normally be used with this button, but without knowing each and every camera model's functions and the creative ways some might use scripts, these two command-methods are also mentioned.

## *click/press/release "erase"*

Actuates the camera's erase button. (Note: actuates the FUNC (function) button for S-series cameras.)

This will often be used to select some shooting parameters like exposure-compensation, movie frame-rates, white-balance settings, ... any of the options that can be reached by

pressing this button on your camera. It is then used in conjunction with directional button-presses to choose the desired settings.

Note: **press** and **release** would not normally be used with this button, but without knowing each and every camera model's functions and the creative ways some might use scripts, these two command-methods are also mentioned.

Example:

```
@title EXP bracketing
@param a Number of +/- steps
@default a 2
@param b Step size (1/3EV)
@default b 3

if a<1 then let a=2
if b<1 then let b=3

sleep 1000

print "Preparing..."
click "erase"
for n=1 to a*b
    click "left"
next n

for s=1 to a*2
    print "Shoot", s, "of", a*2+1
    shoot
    for n=1 to b
        click "right"
    next n
next s

print "Shoot", a*2+1, "of", a*2+1
shoot

print "Finalizing..."
for n=1 to a*b
    click "left"
next n
click "erase"

end
```

In this "Exposure Bracketing" script, if you follow the embedded button-presses, you'll see that your Exposure Compensation setting is being selected by using the **click "erase"** command. The **click "right"** and **click "left"** commands are moving the Exposure compensation settings to the right and left (more exposure and less exposure), just as you would if you were doing this manually from one shot to the next.

## *click/press/release "iso", "flash", "mf", "macro", "video", "timer" (S-series only)*

Actuates the S-series specific buttons.

(This will need to be added to with a few examples, specifically in using the new press/release commands with some of these.)

## *The* wait_click *and* is_key *commands*

Syntax:

**wait_click** ***timeout*** (waits for any button to be clicked; *timeout* is optional)

**is_key x "button-name"** (if last clicked key was **"button-name"** 1 will be placed in variable x; for timeout checking **"no_key"** is used as button name)

Examples

```
...
:wait
  wait_click

  is_key k "set"
    if k=1 then goto "continue"
goto "wait"

:continue
...
...
:loop
  wait_click 5000

  is_key k "left"
    if k=1 then gosub "k_left"
  is_key k "right"
    if k=1 then gosub "k_right"
  is_key k "set"
    if k=1 then goto "k_set"
  is_key k "no_key"
    if k=1 then goto "timeout"
goto "loop"

:timeout
print "Timeout"
goto "end"

:k_left
...
return

:k_right
...
return

:k_set
...
:end
end
```

## *The set_tv, get_tv, etc commands*

There are several commands for getting and setting the aperture and the speed. They only work in Manual mode; well you can change the settings in any mode, but they are effective in manual mode, probably also in Av and Tv modes). I put a test script for these commands in the "user written scripts"

**The commands are**

```
get_tv target
set_tv_rel val
set_tv val
get_av target
set_av_rel val
set_av val
```

Target is the name of a variable (a, b, ..z), val is an expression.

An example of setting and printing the values.

```
:set_get
set_av c
set_tv b
print "AV,TV set to",c,b
sleep 1000
click "shoot_half"
sleep 100
get_av n
get_tv m
print "AV,TV got",n,m
end
```

You can change the settings relative to existing values:

```
rem increase light (1/3+1/3 steps)
set_tv_rel 0-1
set_av_rel 0-1
shoot
end
```

This might make bracketing easier and faster

For A710is the Av and Tv settings provide the following actual values; roughly +/-1 setting means +/-1/3 EV change:

| set_av | | set_tv | | set_tv | | set_tv | |
|---|---|---|---|---|---|---|---|
| 9 | 2.8 | –12 | 15” | 3 | 0.5” | 19 | 1/80 |
| 10 | 3.2 | –11 | 13” | 4 | 0.4” | 20 | 1/100 |
| 11 | 3.5 | –10 | 10” | 5 | 0.3” | 21 | 1/125 |
| 12 | 4.0 | –9 | 8” | 6 | 1/4 | 22 | 1/160 |
| 13 | 4.5 | –8 | 6” | 7 | 1/5 | 23 | 1/200 |
| 14 | 5.0 | –7 | 5” | 8 | 1/6 | 24 | 1/250 |
| 15 | 5.6 | –6 | 4” | 9 | 1/8 | 25 | 1/320 |
| 16 | 6.3 | –5 | 3.2” | 10 | 1/10 | 26 | 1/400 |
| 17 | 7.1 | –4 | 2.5” | 11 | 1/13 | 27 | 1/500 |
| 18 | 8.0 | –3 | 2” | 12 | 1/15 | 28 | 1/640 |
| | | –2 | 1.6” | 13 | 1/20 | 29 | ??? |
| | | –1 | 1.3” | 14 | 1/25 | 30 | 1/800 |
| | | 0 | 1” | 15 | 1/30 | 31 | 1/1000 |
| | | 1 | 0.8” | 16 | 1/40 | 32 | 1/1250 |
| | | 2 | 0.6” | 17 | 1/50 | 33 | 1/1600 |
| | | | | 18 | 1/60 | 34 | 1/2000 |

For A610 the Av and Tv settings provide the following actual values:

| set_av | | set_tv | | set_tv | | set_tv | |
|---|---|---|---|---|---|---|---|
| 9 | 2.8 | –13 | 15” | 3 | 0.5” | 19 | 1/80 |
| 10 | 3.2 | –12 | 13” | 4 | 0.4” | 20 | 1/100 |
| 11 | 3.5 | –11 | 10” | 5 | 0.3” | 21 | 1/125 |
| 12 | 4.0 | –10 | ? | 6 | 1/4 | 22 | 1/160 |
| 13 | 4.5 | –9 | 8” | 7 | 1/5 | 23 | 1/200 |
| 14 | 5.0 | –8 | 6” | 8 | 1/6 | 24 | 1/250 |
| 15 | 5.6 | –7 | 5” | 9 | 1/8 | 25 | 1/320 |
| 16 | 6.3 | –6 | 4” | 10 | 1/10 | 26 | 1/400 |
| 17 | 7.1 | –5 | 3.2” | 11 | 1/13 | 27 | 1/500 |
| 18 | 8.0 | –4 | 2.5” | 12 | 1/15 | 28 | 1/640 |
| | | –3 | 2” | 13 | 1/20 | 29 | ??? |
| | | –2 | 1.6” | 14 | 1/25 | 30 | 1/800 |
| | | –1 | 1.3” | 15 | 1/30 | 31 | 1/1000 |
| | | 0 | 1” | 16 | 1/40 | 32 | 1/1250 |
| | | 1 | 0.8” | 17 | 1/50 | 33 | 1/1600 |
| | | 2 | 0.6” | 18 | 1/60 | 34 | 1/2000 |
| | | | | | | 35 | 1/2500 |

For the S2/S3 IS the Av and Tv settings provide the following actual values:

| Aperture | | Exposure | | | |
|---|---|---|---|---|---|
| **Value** | **index** | **Value** | **index** | **Value** | **index** |
| F/2.7 | 0 | 15” | –12 | 1/15 | 12 |
| F/3.2 | 10 | 13” | –11 | 1/20 | 13 |
| F/3.5 | 11 | 10” | –10 | 1/25 | 14 |
| F/4.0 | 12 | 8” | –9 | 1/30 | 15 |
| F/4.5 | 13 | 6” | –8 | 1/40 | 16 |
| F/5.0 | 14 | 5” | –7 | 1/50 | 17 |
| F/5.6 | 15 | 4” | –6 | 1/60 | 18 |
| F/6.3 | 16 | 3”2 | –5 | 1/80 | 19 |
| F/7.1 | 17 | 2”5 | –4 | 1/100 | 20 |
| F/8.0 | 18 | 2” | –3 | 1/125 | 21 |
| | | 1”6 | –2 | 1/160 | 22 |
| | | 1”3 | –1 | 1/200 | 23 |
| | | 1” | 0 | 1/250 | 24 |
| | | 0”8 | 1 | 1/320 | 25 |
| | | 0”6 | 2 | 1/400 | 26 |
| | | 0”5 | 3 | 1/500 | 27 |
| | | 0”4 | 4 | 1/640 | 28 |
| | | 0”3 | 5 | 1/800 | 30 |
| | | 1/4 | 6 | 1/1000 | 31 |
| | | 1/5 | 7 | 1/1250 | 32 |
| | | 1/6 | 8 | 1/1600 | 33 |
| | | 1/8 | 9 | 1/2000 | 34 |
| | | 1/10 | 10 | 1/2500 | 35 |
| | | 1/13 | 11 | 1/3200 | 0 |

For Tv, 2 values are equal, 1” and 1/3200 with 0.

CHDK Build 119 Remapped Values (* = change):

For S3-IS (should be the same on all camera models now? If so we can remove the previous tables. Please confirm on other models.)

| Aperture | | Exposure | | | |
|---|---|---|---|---|---|
| **Value** | **index** | **Value** | **index** | **Value** | **index** |
| * F/2.7 | 9 | * 15” | –12 | 1/15 | 12 |
| F/3.2 | 10 | * 13” | –11 | 1/20 | 13 |
| F/3.5 | 11 | * 10” | –10 | 1/25 | 14 |
| F/4.0 | 12 | 8” | –9 | 1/30 | 15 |
| F/4.5 | 13 | 6” | –8 | 1/40 | 16 |
| F/5.0 | 14 | 5” | –7 | 1/50 | 17 |
| F/5.6 | 15 | 4” | –6 | 1/60 | 18 |
| F/6.3 | 16 | 3”2 | –5 | 1/80 | 19 |
| F/7.1 | 17 | 2”5 | –4 | 1/100 | 20 |
| F/8.0 | 18 | 2” | –3 | 1/125 | 21 |
| | | 1”6 | –2 | 1/160 | 22 |
| | | 1”3 | –1 | 1/200 | 23 |
| | | 1”“ | 0 | 1/250 | 24 |
| | | 0”8 | 1 | 1/320 | 25 |
| | | 0”6 | 2 | 1/400 | 26 |
| | | 0”5 | 3 | 1/500 | 27 |
| | | 0”4 | 4 | 1/640 | 28 |
| | | 0”3 | 5 | * 1/800 | 29 |
| | | 1/4 | 6 | * 1/1000 | 30 |
| | | 1/5 | 7 | * 1/1250 | 31 |
| | | 1/6 | 8 | * 1/1600 | 32 |
| | | 1/8 | 9 | * 1/2000 | 33 |
| | | 1/10 | 10 | * 1/2500 | 34 |
| | | 1/13 | 11 | * 1/3200 | 35 |

**Usage Notes**

When using the **set_tv**, **set_tv_rel**, or **set_av**, **set_av_rel** commands it was found that it might not be effective if inserted into a sequence of commands that used the **press** and in some instances the **click “button”** commands. If when testing your script you find these commands will not alter the shutter-speed or aperture, try moving them to a position just before any **press “shoot_half/full”** or **click “timer”** (unique s-series) commands. For an example see the “Lightning Photography” scripts for where the **set_tv** command had to be placed before it would work. It was tried in all other locations before the actual shooting was to begin, setting the shutter-speed in other locations in the script wouldn’t work otherwise.

### *The set_zoom, set_zoom_rel, get_zoom, set_zoom_speed commands*

(CHDK Build 119 or greater, command *set_zoom_speed* is available in Build 122 or greater)

Syntax:

**set_zoom x** (where x is 0 to 8, 14, or 129, see Range)

**set_zoom_rel x** (x is +/- relative change)

**get_zoom x** (zoom-step value placed in variable x)

**set_zoom_speed x** (where x can be from 5-100 range. Will do nothing for A-series)

(5 is 5% of high-speed, 100 is 100% of high-speed)

Range:

A-series: x = 0 to 8 or 14 (9 or 15 steps)

S-series: x = 0 to 128 (129 steps)

Note 1: Camera does not refocus automatically after the end of zooming. Use a click or press/release "shoot_half" command to implement a refocusing if needed.

Note 2: It was found that if using the slowest speed (5), that an S3 IS might shut down after it has waited too long for the zoom to traverse the whole range of 129 steps. A speed of 10 did not exhibit this behaviour on an S3 IS. 5 is **so slow** though, that I think it would rarely be needed, except in movie-shooting scripts, and then the range could be limited to prevent camera shut-down.

**Note 3: CAUTION!** (Found on S3 IS) If **set_zoom_speed** is not written into the script when **set_zoom x** is used, the camera will refocus some of your optics to make it where the camera is unable to focus on anything in any mode. The camera (when zooming without a set-zoom speed) appears to move an internal lens element that puts the lens into a Super-Macro mode where it focuses on internal lens elements at widest-angle. If this command is left out of a script using the **set_zoom x** command, you will have to shut down your camera and restart it to reset the zoom-lens' optics back to defaults. However, an interesting thing is found -- when running the "Zoom-Shoot" script by rem-ing out the set_zoom_speed command (removing it from being implemented), after the camera resets its zoom, the lens is now in a ZOOMED tele-macro SUPER-MACRO MODE! Giving you close-up focusing ability at fullest zoom! (As if you had placed a +4 or so close-up lens on your camera.) Far surpassing the capabilities that Canon designed. Perhaps this "bug" could be put to great use? Or it might damage your focusing and zooming mechanisms. USE WITH CAUTION. Because you can hear the camera strain up against some internal lens-adjustment stops when it's trying to reset the zoom. And the only way to "un-do" this (really nice!) tele-super-macro mode is by turning the camera off and on again.

(Leaving this small chart here for reference, but is no longer applicable to the new set_zoom commands.):

S-Series Zoom Speed: 36mm-432mm or 432mm-36mm

slow = 6 seconds (available in single-shot & movie mode)

medium = 4 seconds (available in movie mode)

high-speed = 1 second (available in single-shot mode)

## The set_focus and get_focus commands

(CHDK Build 125 or greater)

Syntax:

**set_focus x** (where x is the distance in mm)

**get_focus x** (the distance value placed in variable x)

## The set_iso and get_iso commands

(CHDK Build 125 or greater)

Syntax:

**set_iso x** [where x is one of the following values: 0 = AutoISO; 1, 2, 3, 4, 5 = 50(80),100,200,400,800; –1 = HiISO (where applicable)]

**get_iso x** (the ISO value placed in variable x)

## Special Build Commands

Due to the open-source sharing of this project, many other talented individuals have been creating their own versions of CHDK, some with exceptional improvements or features that don't exist in the original CHDK. An attempt will be made to include the commands of those builds that have important features worth considering. Please note that any commands that appear in the "Special Builds" sections in this tutorial will not work with the original CHDK by GrAnde, unless he sees fit to include them in his own builds one day.

### MX3's Motion Detection Build

**Note 1**: These uBASIC script commands are available in a special build of CHDK (available here). It is unknown at this time if this will become a standard feature of CHDK or not, so these commands are being placed in their own section so as not confuse people (thinking that these will work with the standard CHDK platform). (This has gained immense popularity and has become a standard feature in nearly all builds of CHDK, including the latest Allbest Build.)

**Note 2**: MX3's Motion-Detection has also been included in Fingalo's and Microfunguy's Special Builds, see their extra commands below.

**Note 3**: There has been much discussion on the proper ways to use this sometimes-confusing and highly adaptable and user-configurable feature. A lengthy discussion on the new CHDK Forum on how to get the fastest reaction times for lightning photography has shed some light on the subject (pun not intended). For further clarification on the best ways to implement some of the timing controls, see this post in the "Motion Detection Too Slow?" discussion thread, which also includes a script optimized to obtain the fastest detection speed possible by using 2 different methods (both available in the same script). The MD routine has been reworked for some cameras so the internal "immediate shoot" option is now lightning-fast (literally). This change will probably be added to all new future builds (note added 2008-02-07 c.e.).

*I am not the author of this feature, so some errors may exist in the information below. Hopefully the author will check in to see if this is all correct or not. The main crux of it being taken from MX3's own demo and test script files.*

**Available Commands**

### md_detect_motion

This command is the main crux of setting all feature parameters.

```
                /--/-COLUMNS, ROWS to split picture into
                |  |  MEASURE MODE (Y,U,V R,G,B) - U=0, Y=1, V=2, R=3, G=4, B=5
                |  |  |  TIMEOUT
                |  |  |  |  COMPARISON INTERVAL (msec)
                |  |  |  |  |  THRESHOLD (difference in cell to trigger detection)
                |  |  |  |  |  |  DRAW GRID (0=no, 1=yes)
                |  |  |  |  |  |  |  RETURN VARIABLE number of cells with motion detected
                |  |  |  |  |  |  |  |           OPTIONAL PARAMETERS:
                |  |  |  |  |  |  |  |  REGION (masking) mode: 0=no regions, 1=include, 2=exclude
                |  |  |  |  |  |  |  |  |
                |  |  |  |  |  |  |  |  |  REGION FIRST COLUMN
                |  |  |  |  |  |  |  |  |  |  REGION FIRST ROW
                |  |  |  |  |  |  |  |  |  |  |  REGION LAST COLUM
                |  |  |  |  |  |  |  |  |  |  |  |  REGION LAST ROW
                |  |  |  |  |  |  |  |  |  |  |  |  |  PARAMETERS: 1=make immediate shoot,
                |  |  |  |  |  |  |  |  |  |  |  |  |  |   2=log debug information into file (* see note below!),
                |  |  |  |  |  |  |  |  |  |  |  |  |  |   4=dump liveview image from RAM to a file,
                |  |  |  |  |  |  |  |  |  |  |  |  |  |   8=on immediate shoot, don't release shutter.
                |  |  |  |  |  |  |  |  |  |  |  |  |  |   OR-ed values are accepted, e.g. use 9 for
                |  |  |  |  |  |  |  |  |  |  |  |  |  |   immediate shoot & don't release shutter
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  PIXELS STEP - Speed vs Accuracy adjustments
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  |   (1=use every pixel,
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  |    2=use every second pixel, etc)
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  MILLISECONDS DELAY to begin triggering
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |   Can be useful for calibration with
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |   DRAW-GRID option.
                |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
md_detect_motion a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p
```

The minimum number of variables that must be set with this command are:

md_detect_motion a, b, c, d, e, f, g, h

**Timeout (d)**: [mx3] is time in milliseconds for which md_detect_motion will block execution of next uBASIC commands, if during this period no motion is detected. This parameter is useful if you want to execute periodically some other uBASIC commands together with MD.

E.g. MD routine waits for changes for 1 second; if no motion detected, script can continue to execute some other code and then, if required, can resume motion detection by again calling md_detect_motion. So timeout is just the time for which MD routine will wait for changes. In practice, this TIMEOUT value (parameter d) should be greater than the MILLISECONDS DELAY (parameter p), or else you will always get RETURN VARIABLE (parameter h) = 0.

**Comparison Interval (e)**: The time delay in milliseconds in which to check for a change in a cell's values. If you need to filter out small changes made frequently by faster moving objects (leaves in the wind, or flying insects, for example) you would increase this value so that timed samples are further apart. Very useful when trying to detect changes in very slow moving subjects, e.g. snails, slime-moulds, a slow-moving criminal trying to avoid motion detection devices ☺, etc.

**h – RETURNED VARIABLE**: this variable is used for deciding whether you want to shoot. It contains a count of cells where the change is more than the specified threshold value.

Example: `if h>0 then shoot`

**n=2 (debug mode)**: Since build #684 (Jan 18th 2009), this debug feature has been removed to save RAM. To use it, a custom CHDK version must now be built (OPT_MD_DEBUG=1 in makefile.inc will enable motion detector debug).

(Insert more information on variable parameter functions and uses as they become known or more familiar.)

**md_get_cell_diff**

[mx3] This is an optional procedure for those people who want to know where in the scene detection actually happened. This procedure is designed for scene change advanced analysis. I'm not sure many people will need to use it. Most people will use the "h" variable from above example to trigger shooting.

Usage: `md_get_cell_diff (column), (row), x`

where x will be difference of 0 to 255 between the last and present change in that cell. Triggering a script to shoot on this value may be done by detecting no change, or however much sensitivity you would like to detect in that cell.

Examples:

If you would like to have the camera shoot an image when all motion stops, use:

```
if x=0 then "shoot"
```

To shoot an image when any motion is detected at all use:

```
if x>0 then "shoot"
```

Interesting use of MD:

The following was copied from a post where MX3 mentions a feature of md_get_cell_diff that was never documented before.

*Nobody tried to use MD to get overall luminosity to automatically adjust shutter speed override?*

*MD setup:*

*set delay_interval to 2–3 secs*
*timeout=delay_interval+1*
*threshold=255 (so it will not trigger)*
*cols=1*
*rows=1*
*md_get_cell_diff 1, 1, overall_luminosity*
*shutter_override_time = some_formula(overall_luminosity)*

*I don't have camera nearby to test it.*

*I have thought about time-lapse movie script that would automatically override shutter speed at night. I'm planning to make 2 days time-lapse movie (it seems 8GB SD card and power adapter will help also ☺)*

***NOTE:** when MD stops working on timeout, cells contain absolute values instead of difference.*

The most important info is contained in that final "NOTE"!

Referring to the 'md_detect_motion' command-parameters in the WIKI article, '**a**' and '**b**' define the number of rows and columns to split the screen into. (If values less than zero are entered or if total number of cells is greater than 1024, it defaults to 3 x 3.)

Parameter '**g**' determines if the grid showing the detected cells is displayed.

Parameters '**j**, **k**, **l**, **m**' define a sub-area of the screen where motion-detection is restricted-to or excluded-from.

Parameter '**i**' determines if the region is inclusion/exclusion or do not use regions.

You may detect motion based on changes of luminance (Y), blue-luminance (U), red-luminance (V) or individual R, G or B values.

Parameter '**c**' sets that mode.

(For an example of an image split into it's YUV components, see the WIKI article.)

For non-specialised use, luminance (c = 1) will be used.

You then need to set a threshold-value (in parameter '**f**') for the desired mode that will not result in triggering in 'normal' operation.

The motion-detection event may be triggered by quick or slow changes in the screen image; set a suitable value with parameter '**e**'.

The greatest accuracy of movement-detection results when every pixel is sampled, but a faster response (suitable for some applications) may be obtained with a larger pixel-step.

Set an appropriate value in parameter '**o**'.

Set a maximum-time for a motion-detection event to occur with parameter '**d**' so that after that time the script-command terminates.

**Motion-detection Parameters:**

columns, input parameter. Number of columns to split screen into

rows, input parameter. Number of rows to split screen into

pixel_measure_mode, input parameter. 1 for Y, 2 for U, 3 for V, 4 for gray, 5 for R, 6 for G, 7 for B

detection_timeout, input parameter. Number of milliseconds to abort detection. detected_cells_count will be 0 for timeout condition

measure_interval, input parameter. Number of milliseconds between comparison of two pictures

threshold, input parameter. Difference value for which procedure will trigger detection of changes

draw_grid, Boolean input parameter. True (1) to draw grid(detected sectors/cells). False (0) to not display grid/detected sectors

detected_cells_count, output parameter. Count of cells where pixel values differs enough to trigger motion detection

clipping, allows to exclude some region from motion detection triggering, or use only selected area to make motion detection

I'm not sure that following parameters are required but using them anyway

clipping_region_mode, input parameter. 0 = no clipping regions, 1 = excludes selected region from motion detection, 2 = use only this region to make motion detection

clipping_region_column1, input parameter.

clipping_region_row1, input parameter. This is top-left corner of clipping region

clipping_region_column2, input parameter.

clipping_region_row2, input parameter. This is right bottom corner of clipping region)

function md_get_cell_diff (col [in] = column of the cell we are requesting, row [in] = row of the cell we are requesting, val [out] = value of difference between measurements/ comparisons)

Reserved parameters clipping regions, pixel_measure_mode, draw_grid

## Fingalo's Builds

Available from: Fingalo's CHDK2

Added commands: **SET_LED**, **GET_VBATT**, **SET_RAW**, and **SET_PROP** / **GET_PROP**

Used to control the external LED lamps, read the battery voltage, turn RAW image recording on and off from scripts, and to set/read "property-case" values (respectively).

**NOTE**: Fingalo has also included the amazing Motion-Detection command from MX3 as outlined above. See MX3's Motion Detection Commands

### LED Lamp Control (Fingalo's builds only)

Usage:

**set_led a b c**

Fingalo says, “ONLY for S3 (and S2 I guess)”

I tried with A560 and it worked, but there is no LED 10 (and no 6). And I think these work with any A500-series camera.

Parameter **a** is the LED-lamp as follows:

| a | LED Lamp |
|---|---|
| 4 | GREEN (by power switch on S3 and A560) |
| 5 | YELLOW (by power switch on S3 and under green LED on A560 ) |
| 6 | (not used) |
| 7 | ORANGE (red LED on back of S3 and same place than green on A560) |
| 8 | BLUE |
| 9 | Focus Assist/Auto-Focus Lamp/AF Lamp (bright green on S3 & bright orange on A560) |
| 10 | Timer/Tally Lamp (bright orange lamp in front on S3) |

Parameter **b**

0 LED is off, 1 LED is on

Parameter **c** (optional) is brightness

0–200, (Fingalo says, “Seems to work only on the blue LED.”)

(LEDs work on A560, but brightness doesn’t work for any of them.)

Example:

```
rem Turn on AF_Lamp, Focus Assist Lamp
set_led 9 1

rem Turn on Blue LED with reduced brightness
set_led 8 1 35
```

**IMPORTANT NOTE**: When using any LED lamp controls, remember to reset them to their original condition as they were before executing your script. Failure to do so may result in your power-indicator not alerting you that your camera still powered on. Or other important camera functions involving the LED lamps may not light at their proper times.

Note 2: When testing the Blue LED brightness by putting it in a for x=0 to 200 loop to ramp the value all the way up in 1 value increments, then and back down again, it doesn’t appear to behave linearly. The LED ramps up, then turns off, briefly flashes, ramps up again, flashes, then ramps down and flashes (or something similar to that). I suspect it might be working from 0 to 127 using binary bit values. But I’ve not tested it for this.

### GET_VBATT Read Battery Voltage

Read voltage, momentary value, varies a bit.

Usage:

a = **get_vbatt**

Value is returned in mV (millivolts, 1/1000$^{th}$ of a volt).

### SET_RAW enable/disable RAW Recording

(Fingalo's builds only [AllBest too])

Usage:

**set_raw** a

where: a = 0 RAW recording is **OFF**; a = 1 RAW recording is **ON**

### MORE USER VARIABLES!!!!!!

(Currently, Fingalo's v106 build only)

Fingalo's version 106 allows UPPERCASE variables, in addition to lowercase variables. You can use a-z and A-Z, for a total of 52 unique variables!

### SET Dark Frame subtraction state (ON|OFF|AUTO)

(Currently only in Fingalo's v106 or later)

Determines whether the camera will do a dark frame subtraction after taking a shot. Auto means the camera decides, OFF means no, ON means yes. Dark frame acquisition and subtraction typically occurs for images with an exposure time of 2/3 of a second or longer (1.3 sec for A470?). It does consume time (it's equivalent to taking another image at the same exposure time).

Note: although this command refers to "raw", it actually applies regardless of whether you are in RAW mode or not. AUTO is the state the camera normally is in. CHDK allows you to change this to the ON or OFF states, and this uBASIC command allows you to change it in a script.

Usage:

**set_raw_nr a**

where the variable **a** determines the state: 0=Auto, 1=OFF, 2=ON

### get_tick_count

(Not in GrAnd August 2007 versions)

This function returns the time, in milliseconds, since the camera was turned on. Note that this function format is a bit different from the standard CHDK uBASIC function format.

Usage:

**t = get_tick_count**

### get_day_seconds

(Not in GrAnd Aug 07 versions)

This function returns the number of seconds since midnight. Note that this function format is a bit different from the standard CHDK uBASIC function format.

Usage:

**t = get_day_seconds**

For a simple example using this function to wait until a specific time of day before continuing, see get_day_seconds_example.

## SET/GET_PROP – Read/Set Property-Case Values

(Fingalo's builds only)

This is a powerful pair of commands. These are used to read and set "property-case" values in the firmware of your camera. They can be used for: detecting and setting the flash mode, mode-dial position, the internal self-timer delay, video frame rates, and more.

**A new page has been created to describe the use of some of the more useful property case values. See this link The Property Case Use page**

The presently known property-case values were originally taken from a list posted at a Russian authored List of known Property Cases. A more up-to-date list can be found here: this page of Property Case IDs. [There is now a Discussion page section for user contributions to determining the values and uses of the property cases. It also has a link to scripts for exploring these items. You can find it here: Property case exploration.]

**IMPORTANT**

USE THE SET_PROP COMMAND WITH CAUTION. NOT ALL HAVE BEEN TESTED FOR POSSIBLE OUTCOMES.

Usage:

**set_prop** propid value
**get_prop** propid value

where **propid** may be any of the following:

| PropID | Description |
|---|---|
| 0 | Shooting mode dial position |
| 1 | Photo effect |
| 5 | White balance |
| 6 | Drive mode (S3 values: 0 = single, 1 = continuous, 2 = timer) |
| 8 | Hi-speed continuous mode (S3: 1 = OFF, 0 = ON |
| 9 | Metering mode (S3 values: 0 = eval 1 = spot 2 = centre) |
| 11 | Macro (S3 values: 0 = normal, 1 = macro, 2 = super macro) |
| 12 | Manual Focus (S3 values: 1 = manual, 0 = auto) |
| 14 | Delay of self-timer (appears to be time in milliseconds) |
| 16 | Flash mode (S3: 2 = flash closed, otherwise 0 = auto, 1 = ON) |
| 18 | Red eye mode (S3: 0 = OFF, 1 = ON) |
| 19 | Flash slow sync (S3: 0 = OFF, 1 = ON) |
| 20 | Flash Sync Curtain (S3: 0 = first, 1 = second) |
| 21 | ISO value (S3: 0 = auto, 1 = ISO-HI, or actual ISO: 80,100,200,400,800) |
| 23 | Image quality (S3 values: 0, 1, 2 from best to worst) |
| 24 | Image resolution (S3 values: 0, 1, 2, 4, 8 for L, M1, M2, S, W) |
| 25, 26 | EV correction (positive or negative, 96 units per stop) |
| 28 | Flash correction (same units as 25/26) |

| PropID | Description |
|---|---|
| 32 | Exp bracket range (Same units as 25/26: e.g. 96 = +/- 1 stop range) |
| 34 | Focus bracket range 2 = Smallest, 1 = Medium, 0 = largest |
| 36 | Bracket mode: 0 = NONE, 1 = exposure, 2 = focus |
| 37 | Orientation sensor |
| 39 | Chosen Av (by user) |
| 40 | Chosen Tv (by user) |
| 65 | Focus distance |
| 67 | Focus OK: 1 = Yes, 0 = NO |
| 68 | Coming Av |
| 69 | Coming Tv |
| 74 | AE lock: 1 = ON, 0 = OFF |
| 126 | Video FPS (15, 30 or 60. Don't change here!) |
| 127, 128 | Video resolution (S3: 2, 1 for 640x480; 1, 0 for 320x240) |
| 177 | Intervalometer: #of shots (0 if not activated) |
| 205 | ? '1' during shooting process |
| 206 | "MyColors?" mode (see link below) |
| 218 | Custom timer continuous: # of shots to be taken |
| 219 | Self-Timer setting: 0 = 2 sec, 1 = 10 sec, 2 = custom/continuous |

And **value** may be any that is appropriate for that particular **propid**.

2, 3, 4, 207, 208, 209, 210 contain individual parameters for the "Custom" MyColors setting

Additional information (hopefully growing) about what values might work for some of these properties can be found at the following link: Property case exploration page. This link also has a more complete description of the MyColors settings (contrast, saturation, sharpness, individual color intensities, etc)

Example script for setting and viewing Prop_IDs.

```
@title popcase
@param a propid
@default a 0
@param b value
@default b 0
:loop
  wait_click
  is_key k "left"
  if k=1 then set_prop a b
  is_key k "set"
  if k=1 then goto "lend"
  get_prop a b
  print a,b
goto "loop"
:lend
end
```

## Allbest's Builds

The Allbest build is a major rewrite of CHDK, in many ways. It also includes many new uBASIC commands. Below is a partial list of the complete list of available commands, those which are unique to the Allbest build. These have not been documented in total yet, and more uBASIC commands are being added frequently. (Please see the CHDK forum for discussions of any works in progress.)

NOTE: Syntax usage in most cases is **command_name x**, where x either sets or returns the value in that command. Unless stated otherwise, assume this usage syntax. Otherwise they may be acting as their own variable, and may be used as-is in a command string. Example: get_vbatt is its own variable. It can either be assigned to another variable with **x=get_vbatt**, or used on its own as in **print get_vbatt**. The different types of uBASIC command syntax will be clarified as needed or as discovered. (Developers don't document things very well. We, as end-users, sometimes have to find these things by trial-and-error, or be perceived as a major nuisance by hounding them for any clues into what they did. ☺ I use both methods. ☺)

**Get ops commands (to be associated with suitable return parameters):**

"get_av96"

*Since Canon appears to use ?? to calculate the rate of 96 (found by analysing the values match formulas APEX), and also in the token indicates that the installation corresponds to the value of Av * 96. From my point of view, so convenient. Load-meaning teams receive appropriate value in a variable Av * 96 from the relevant again, PropertyCase*.
(This wording is an online automated translation from the original info in German. If anyone would like to make sense of what it's saying, please correct this text. {I'll have a crack if someone points me to the original German text [DF]})

Example of usage (set_shutter for Ixus) by Allbest:

```
@title Shutter TEST
sleep 500
rem initiation
press "shoot_half"
release "shoot_half"
get_tv96 t

:set_shutter
  print "Tv set to",t
  wait_click
  is_key k "set"
  if k=1 then goto "k_set"
  is_key k "down"
  if k=1 then t=t-32
  k=0
  is_key k "up"
  if k=1 then t=t+32
  k=0
  set_tv96_direct t
  goto "set_shutter"

:k_set
  shoot
  end
```

"get_bv96" get brightness value

"get_day_second current within one second of the day

Syntax: x=get_day_seconds

get_day_seconds acts as its own variable, This allows you to even use it within calculations without first assigning it to another variable.

"get_dof" get the depth of sharpness in mm

"get_far_limit" get the border zone ranged acceptable sharpness mm

"get_focus"

"get_hyp_dist" get hyperfocal distance

"get_iso_market" get "marketing" ISO (See the Allbest's Firmware Usage page on ISO values for what is meant by a "Market Value".)

"get_iso_mode" obtain ISO mode (the former get_iso) ⟶ E.g. the A620 ISO list:

| | |
|---|---|
| 0 | Auto |
| 50 | 50 |
| 100 | 100 |
| 200 | 200 |
| 400 | 400 |

"get_iso_real" get real "value" ISO

"get_iso" obtain ISO mode

"get_near_limit" get dipped border zone acceptable sharpness

"get_prop" obtain property case, call

"get_sv96, receive sensitivity value in the standard APEX (Additive system of Photographic Exposure, see http://en.wikipedia.org/wiki/APEX_system). As always, multiplied by 96

"get_tick_count" returns system time

Syntax: x=get_tick_count

get_tick_count acts as its own variable. This allows you to even use it within calculations without first assigning it to another variable.

"get_tv96, tv * 96

"get_user_av_id", the former get_av. Get custom installation av (in the manual modes) for ID in CHDK:

E.g. the A620 list:

| av_ID | av * 96 | Aperture |
|---|---|---|
| 9 | 288 | f2.8 |
| 10 | 320 | f3.2 |
| 11 | 352 | f3.5 |
| 12 | 384 | f4.0 |
| 13 | 416 | f4.5 |
| 14 | 448 | f5.0 |
| 15 | 480 | f5.6 |
| 16 | 512 | f6.3 |
| 17 | 544 | f7.1 |
| 18 | 576 | f8.0 |

(The step between successive IDs is a shift of $^{1}/_{3}$ EV)

“get_user_av96” returns custom av * 96

“get_user_tv_id” returns CHDK identifier for the established user manual modes tv

E.g. the A620 list:

| | | |
|---|---|---|
| -4 | -128 | “2.5” |
| -3 | -96 | “2” |
| -2 | -64 | “1.6” |
| -1 | -32 | “1.3” |
| 0 | 0 | “1” |
| 1 | 32 | “0.8” |
| 2 | 64 | “0.6” |
| 3 | 96 | “0.5” |
| 4 | 128 | “0.4” |

**Important:** earlier scripts just use the “get_tv” and “get_av” commands, these must be changed to this newer “get_user_tv_id” and “get_user_av_id” commands to make them work properly if using Allbest builds.

This is part of possible values. Meaning load deflection on the id, the same as in the case of av. *(I hope that makes sense to you – means nothing to me! [DF])*

“get_user_tv96” returns value installed in the user manual modes. Important: tv * 96

“get_vbatt”, the voltage of the battery

Syntax: x=get_vbatt

get_vbatt acts as its own variable. This allows you to use it even within expressions, e.g. `if (get_vbatt <= 4300) then print “DEAD BATTERY!”`

“get_zoom”

**Set OPS (usually associated with suitable parameters):**

“set_av96_direct” direct installation av * 96. It works similarly to direct the installation of the interface chdk av. In any mode

“set_av_rel” see “set_user_av_by_id_rel (compatibility)

“set_av96”, the installation of av * 96 in accordance with acceptable Canon list for the camera. Works in any mode

“set_av” see “set_user_av_by_id (compatibility)

“set_focus”

“set_iso_mode”, the installation of an ISO regime

“set_iso_real” Direct installation ISO. It works similarly to the installation of the CHDK ISO interface

“set_iso” – see “set_iso_mode (compatibility)

“set_led” Transmit three-parameters ID, indicator state, and brightness (see p.31/32)

“set_prop”, install PropertyCase (?)

“set_raw_nr”, install script regime for noise reduction: “Auto”, “Off”, “On” (= 0, 1, 2 resp.)

“set_raw”, the installation script in raw mode; disables the last (previous?)

“set_sv96” direct installation of the sensitivity of APEX (Sv * 96)

“set_tv96_direct” direct tv * 96 installation. Works by installing excerpts from the CHDK interface

“set_tv_rel” see “set_user_tv_rel_by_id” (interoperability)

“set_tv96” direct tv * 96 installation from a list of valid CANON values (this value type N * 32. N for the A620 can have values from -12 to 32. Works by installing excerpts from the CHDK interface

“set_tv” see “set_user_tv_by_id” (interoperability)

“set_user_av_by_id_rel” av installation on the current user on bias. The offset indicated in Id. The Id rationale was listed above.

“set_user_av_by_id” custom install av according to the ID in CHDK

“set_user_av96” custom install av * 96 in the manual modes

“set_user_tv_rel_by_id” custom install tv relative to the current tv. The offset is indicated in ID. ID rationale was listed above

“set_user_tv_by_id”, the installation of custom tv permissible, in accordance with Canon ID for CHDK

“set_user_tv96”, the installation of custom Tv * 96. Of the number of allowable and non C

“wheel_right”

“wheel_left”

“get_autostart” parameter checking autostart for scripts

> Syntax: x=get_autostart (or used as it’s own variable-string in calculations; see get_vbatt example)

“set_autostart” Setting this option to autostart scripts

> With this command you should be cautious. Specifying Autorun causes the script to run when you turn the camera on.

“get_usb_power, checking for USB connectivity. Works for series A and S-as a minimum.

> Syntax x=get_usb_power
>
> For G-series is not working. Integration with USB button. (?)

“exit_alt”

**And some recently introduced commands:**

shut_down

> Simply powers-down the camera. Useful for Remote USB scripts where the USB signal may wake up the camera, execute some script function, and then shut down the camera again when done, to save on power for lengthy remote-shooting needs.

Example, `if x=(some calculation) then shut_down`, or just used as a line on its own at the end of your script.

get_disk_size

get_free_disk_space

Returns values in KB. You can build scripts now that stop when a specific disk limit is exceeded. For easier calculation divide by 1024 to return value in MB.

Syntax: **x=get_disk_size**, **x=get_free_disk_space**

Example, to print the space left in megabytes, `print get_free_disk_space/1024` (this, amongst others, is one of those commands that acts as its own variable)

get_jpg_count

get_raw_count

Syntax: **x=get_jpg_count**, **x=get_raw_count** (acts as its own variable which may be assigned to other variables)

Returns the calculated value of how many JPG or RAW shot space is left available on the SD card. (JPG value is approximated and taken from an average of file-sizes, using Canon's own algorithm, the same as shots remaining left in your EVF/LCD display.) Use this command to detect when not enough space is remaining for your required script task to either end the script or shut_down the camera.

set_nd_filter x

Where **x** is

0 = OFF

1 = ND filter IN

2 = ND filter OUT

Added ability to set ND-filter for next set of cameras: a560, a570, g7, ixus700_sd500, ixus70_sd1000, ixus800_sd700, a710 (deeply tested for Ixus800_sd700). This ability replaces aperture override menu entry for Ixus and a560 camera set. For all others from above-mentioned list it is an experimental feature.

get_raw_nr

Returns the condition of your NR (noise reduction setting).

Syntax: **x=get_raw_nr**

### Microfunguy's SDM (StereoData Maker) Builds

Available from: http://stereo.jpn.org/eng/sdm/index.htm

The basic CHDK commands that SDM supports and its own additional 'plain English' commands for custom-timer and continuous-shooting bracketing are detailed at http://stereo.jpn.org/eng/sdm/uBASIC.htm .

The time_lapse command includes options for auto-shutdown, USB stop/start (ideal for KAP), screen blanking and combined Tv and focus bracketing.

The script parameters are described at http://stereo.jpn.org/eng/sdm/tlapse.htm

A number of walk-through examples are at http://stereo.jpn.org/eng/sdm/tlapse2.htm

This example starts at an initial focus position and takes multiple photos at gradually increasing step-size until infinity is reached :

```
set_focus_to 1000
auto_focus_bracketing
" Autofocus bracketing"
" Press switch"
wait_for_switch_press
start_continuous_sequence
wait_until_done
end_continuous_sequence
```

### USB Remote Cable-Release Function!

This amazing feature was found by a talented Ukrainian programmer known as Zosim. You may find his original source code and executable binaries for the A710 IS camera at CHDK binaries and source and photos to build simple cable-release switch. Fingalo and Microfunguy have both added this remarkable addition to their builds of CHDK.

Be SURE to also check out the **Special Builds Features** in the Firmware Usage page for two new Script Menu items on how you can use this feature to completely operate your camera by remote control only. From turning it on to executing your last loaded/used script.

Using nothing more than a 3-volt button-battery and a small switch, you may turn any USB extension cable into a remote shutter release by running a small script. Or by using this script method as a subroutine within your own much more complex scripts.

Most all the cameras are supported for the Remote USB feature, and most all of the different builds also include it now.

There are now two new functions available in the Scripting Menu to enable or disable remote sensing so that you may still download images from your camera while CHDK is still loaded and running as well as a setting to allow activating the last loaded script. See the "Special Build Usage" section for a little more info.

Usage:

special **remote** camera button, used in **is_key** commands with **is_key x "remote"**

Running this small script (or the loop embedded as a subroutine in more lengthy scripts) is all you will need:

```
@title Remote button
:loop
wait_click 1
is_key k "remote"
if k=1 then shoot
goto "loop"
end
```

Or, if using Fingalo's builds you may like his version with the simpler **while/wend** loop commands:

```
@title Remote button
while 1
  wait_click 1
  if is_key "remote" then shoot
wend

end
```

There are many ways of using this "remote" key function; these are just two of the simpler (and faster) ways to implement it.

That's it! That's all you need! Well, one of those little scripts, the right CHDK build, and the cable-switch too. ☺

Between MX3's Motion-Detection options and this amazing USB cable-release method, there is no limit to the various ways you may control your camera by remote means. Any simple electronic circuit that can close a switch and feed a 3v to 5v DC signal to the USB port's proper contacts (observe proper polarity!) can now be used. There is also no limit to the length of wire that you may use, as long as you keep the final contact voltage at the camera-end between the 3vdc and 5vdc range. Use sound-sensitive circuits to record when sound-events happen. Use light or motion changing events to trigger shooting sessions. Use any CHDK intervalometer scripts or electronic intervalometer circuits to trigger shots. (There are thousands of simple circuits like these all over the Internet.) Have your mouse or cat press a switch to record their vanity-quotient for a science-fair project! The sky (literally) is the limit to how many ways you may use these functions.

Have fun!

# Debugging: the Unk Alert

This tiny version of uBASIC includes some debugging help. When running a script with a bad command you might sometimes get a **uBASIC:nn err** statement printed in red in the top-left corner of your EVF or LCD display. This will alert you to why your coding didn't work, albeit in a very abbreviated format giving the line (nn) and error message.

Some examples of what you might see, and what they will mean:

**uBASIC:24 Unk label** Line 24 Unknown label

**uBASIC:32 Parse err** Line 32 Parse error – syntax error in uBASIC command

> See the following sections for IDE and debugging aids ideal for both novice uBASIC developers as well as the more experienced.

## *Debugging Scripts on a PC or Mac*

There are now two ways you can test your CHDK scripts without needing to load them into the camera every time, finding the error and then changing a line, loading it into the camera again and again. The first way is to use the uBASIC_test program, a simple batch program which only runs under Windows. The second way is to use the UBDebug program which runs under Windows or Mac OSX.

### Using UBDebug – an Integrated Development Environment for Scripts

There's now an interactive development environment for uBasic scripts. Written in java with native support for both Windows and Mac OSX it lets you load a script and step through it line-by-line, inspecting and setting variables. You can also set the values to be returned by functions (such as get_usb_power) and alter the value of properties. A simple breakpoint mechanism is available. Scripts can be edited and saved to disk. For details see here

### Using the UBASIC_TEST.EXE Console

Download this small file *uBASIC_test.rar* (if you can find it on a 'clean' website! [DF]), UnRAR (like UnZIP) it to your scripts working location on your hard-drive. You should have

a file named uBASIC_test.exe in your scripts-work folder now. You have to run this program from a Windows Command Prompt (the old time DOS window). Some people have a "Launch Command Prompt Here" on the right-click menu of Windows Explorer, so you can just right-click on the folder where your scripts and uBASIC_test.exe file reside. (You can get this by installing "Open Command Window Here" Power Toy, available here.) Or you can go to Programs > Accessories > Command Prompt (where I have mine for some reason). And use the CD command to Change Directories until you get to where your scripts and uBASIC_test.exe file reside. For example, if you start out in root directory C:\ and your scripts are on drive D: in a sub-folder called CHDK\Files\Scripts\, at the command prompt just type

```
cd D:\CHDK\Files\Scripts
```

and you'll be where you're supposed to be. (You might want to rename that little program to just test.exe to make it easier to type each time.)

To test one of your scripts in that folder, at the Command Prompt, just type "uBASIC_test scriptname.bas" (without the quotes). Where "scriptname.bas" is the name of the script you want to test. It will use the default settings you have assigned to your variables. For testing you should change some of those values to make sure everything is working properly under new user-defined settings. (The reason I suggest you rename that uBASIC_test.exe to just text.exe, is then all you have to type is "test scriptname.bas", saving you a few key-presses.)

The easiest way to run console programs is to use a file manager which has a command line. For example, Far Manager or Total Commander.

You can also test your scripts via drag-&-drop with a batch file. Here's how to do it:

Open a text editor and enter the following lines:

```
@uBASIC_test.exe %1
@pause
```

Save this as "uBASIC_test.bat" in the same folder where your uBASIC_test.exe is. Now you can drag a script with your mouse onto this batch file and it will be executed. (This would also work without making a special batch file, but we need the "pause" command to read the output).

You may need to modify your BAT file to have the **@uBASIC_test.exe %1** line to include the full path to your uBASIC_text.exe file, as well as enclosing the variable **%1** in quotes, in case your script's filename includes any spaces. For example:

```
@H:\Tests\CHDK_Files\SCRIPTS\uBASIC_test.exe "%1"
@pause
```

If you run into problems and this still doesn't work (using this drag & drop method):

1) Make sure your uBASIC_test.exe file and scripts are not in any path that contains spaces. (Example: you can't have it in a sub-folder path of "D:\CHDK Files\Script Tests\uBASIC_test.exe". Change those spaces to _ [underscores] in your actual folder-names if need be.) [DF:] Actually, you **can** have spaces in your path – just enclose the entire path in quotation marks, like so:
   `@"H:\Tests\CHDK Files\SCRIPTS\uBASIC_test.exe"`

2) Your BAT file association may have become corrupted. Here's a handy page of Windows® XP File Association Fixes Get the one for Batch Files. (Save them all, they may come in handy one day!)

(How did I find this out? I had all these problems occurring ☺)

An alternative drag-and-drop method (WinXP):

1) Right-click on uBasic.exe and make a shortcut on desktop,
2) Find/search for your script.
3) drag your script to uBASIC icon, let go and it runs!

You may have to adjust the 'Icon' properties to keep the result on-screen

The addition of a few extra print and rem statements will help debugging, also include values to replace the @defaults.

# Script-Writer's Handy Command-Reference List

I got tired of trying to remember all the commands, so I put together this handy reference list to keep open in my text-editor alongside any scripts I might be working on. I thought it might help other scriptwriters too.

## *CHDK Command List*

(Build 129 or later)

**shoot**

**click "button-name"**

**press "button-name" (used in conjunction with release)**

**release "button-name"**

button-names:

**up / down / left / right**

**set**

**shoot_half**

**shoot_full**

**zoom_in / zoom_out**

**menu**

**display**

**print (means "shortcut" in s-series)**

**erase (means "func"in s-series)**

S-series specific button-names:

**iso**

**flash**

**mf**

**macro**

**video**

**timer**

print commands:

**print "text text text", variable; "text"**

The "print_screen" Command

Whatever the script prints on the mini-console screen is also written to file '/CHDK/SCRIPTS/PR_SCREEN.TXT'.

First call is either:

**print_screen 0** The text is appended to the last file. If the file was there already, the text is written at the end and the older text is not removed.

**print_screen 1** The text is written to "A/CHDK/BOOKS/PS00000.TXT". The new text overwrites any existing text in the file if there was any.

**print_screen N** The text is written to the next file number. The file number cycles between 0 and N-1. If the resulting file number is 5, then the text is written to file "A/CHDK/BOOKS/PS00005.TXT".

The file number of the last written file is kept in file "A/CHDK/BOOKS/PS_COUNT.TXT". Delete the file to reset the counter.

**print_screen 0** turns off writing to the file and **print_screen 1** turns it back on.

Example:

```
@title printscreen test

@param a None
@default a 0

@param c mode: 0-append, 1-replace, other-modulo c
@default c 1

print_screen c
print "START "c
print_screen 0
print "Not written to file"
print_screen 1
print "This should be written to the file."
print "a="a
print_screen 0
end
```

The "cls" Command

**cls** stands for "Clear Screen". This is used to clear the mini-console screen from any "print" statements in an easy way.

**Other commands:**

set_zoom, set_zoom_rel, get_zoom

syntax: set_zoom x (where x is 0 to 8, 14, or 129)

set_zoom_rel x (x is +/- relative change)

get_zoom x (zoom value placed in variable x)

range: A-series: x = 0 to 8 or 14 (9 or 15 steps)

S-series: x = 0 to 128 (129 steps)

**Zoom command restrictions:**

* Camera does not refocus automatically after the end of zooming.
  Use a click or press/release "shoot_half" command to implement a refocusing if needed.
* The "sleep" command is needed after the "set_zoom" command.
  Otherwise, camera will shutdown if other command is executed during zooming process.

set_tv, set_tv_rel get_tv

syntax: set_tv x (where x is the index value)

set_tv_rel x (x is +/- relative change)

get_tv x (index value placed in variable x)

| Exposure | | | | | |
|---|---|---|---|---|---|
| **Value** | **Index** | **(w/ black-frame)** | | **Value** | **Index** |
| 15” | –12 | (~33”) | | 1/15 | 12 |
| 13” | –11 | (~27”) | | 1/20 | 13 |
| 10” | –10 | (~21”) | | 1/25 | 14 |
| 8” | –9 | (~17”) | | 1/30 | 15 |
| 6” | –8 | (~13”) | | 1/40 | 16 |
| 5” | –7 | (~11”) | | 1/50 | 17 |
| 4” | –6 | (~9”) | | 1/60 | 18 |
| 3”2 | –5 | (~7”) | | 1/80 | 19 |
| 2”5 | –4 | (~6”) | | 1/100 | 20 |
| 2” | –3 | (~5”) | | 1/125 | 21 |
| 1”6 | –2 | (~4”) | | 1/160 | 22 |
| 1”3 | –1 | (~3”) | | 1/200 | 23 |
| 1” | 0 | | | 1/250 | 24 |
| 0”8 | 1 | | | 1/320 | 25 |
| 0”6 | 2 | | | 1/400 | 26 |
| 0”5 | 3 | | | 1/500 | 27 |
| 0”4 | 4 | | | 1/640 | 28 |
| 0”3 | 5 | | | 1/800 | 29 |
| 1/4 | 6 | | | 1/1000 | 30 |
| 1/5 | 7 | | | 1/1250 | 31 |
| 1/6 | 8 | | | 1/1600 | 32 |
| 1/8 | 9 | | | 1/2000 | 33 |
| 1/10 | 10 | | | 1/2500 | 34 |
| 1/13 | 11 | | | 1/3200 | 35 |

(S-series)

(note: the w/ black-frame times are approximations for true total-time needed for the longer shutter speeds)

set_av, set_av_rel, get_av

syntax: set_av x (where x is the index value)

set_av_rel x (x is +/- relative change)

get_av x (index value placed in variable x)

| Aperture | |
|---|---|
| **Value** | **Index** |
| f2.7 | 9 |
| f3.2 | 10 |
| f3.5 | 11 |
| f4.0 | 12 |
| f4.5 | 13 |
| f5.0 | 14 |
| f5.6 | 15 |
| f6.3 | 16 |
| f7.1 | 17 |
| f8.0 | 18 |

| ISO | |
|---|---|
| **Value** | **index** |
| AutoISO | 0 |
| 50(80) | 1 |
| 100 | 2 |
| 200 | 3 |
| 400 | 4 |
| 800 | 5 (where applicable) |
| HiISO | –1 (where applicable) |

set_focus, get_focus

syntax: set_focus x (where x is the distance in mm)

get_focus x (the distance value placed in variable x)

set_iso, get_iso

syntax: set_iso x (where x is index value)

get_iso x (index value placed in variable x)

wait_click, is_key

syntax: wait_click (waits for keypress)

is_key k "<key>" (sets k = 1 if the last key pressed was <key>)

**LED Commands:** (Fingalo's Builds)

set_led a,b,c (ONLY for S3 (and S2 I guess))

Parameter **a** is the LED as follows:

| a | LED Lamp |
|---|---|
| 4 | GREEN (by power switch on S3 and A560) |
| 5 | YELLOW (by power switch on S3 and under green LED on A560 ) |
| 6 | (not used) |
| 7 | ORANGE (red LED on back of S3 and same place than green on A560) |
| 8 | BLUE |
| 9 | Focus Assist/Auto-Focus Lamp/AF Lamp (bright green on S3 & bright orange on A560) |
| 10 | Timer/Tally Lamp (bright orange lamp in front on S3) |

Parameter **b**:

0 LED is off
1 LED is on

Parameter **c** (optional):

Brightness 0–200 (seems to work only on the blue LED)

Examples:

```
set_led 9,1        Turn on AF_beam
set_led 8,1,35     Turn on blue LED with reduced brightness
```

**SDM Builds:**

af_led_on
af_led_off

timer_led_on
timer_led_off

blue_led_on
blue_led_off

yellow_led_on
yellow_led_off

green_led_on
green_led_off

amber_led_on
amber_led_off

**Special Build Commands:**

set_prop and get_prop – Set and get property-case values directly. **Use with caution!**

Usage: set_prop a b
where **a** is the property-case location, **b** is the value

## Property Cases:

| Prop_ID | Description |
|---|---|
| 0 | Shooting mode dial position |
| 1 | Photo effect |
| 5 | White balance |
| 6 | Drive mode (S3 values: 0=single, 1=continuous, 2=timer) |
| 8 | Hi-speed continuous mode (S3: 1=OFF, 0=ON |
| 9 | Metering mode (S3 values: 0=eval 1=spot 2=center) |
| 11 | Macro (S3 values: 0=normal, 1=macro, 2=super mac) |
| 12 | Manual Focus (S3 values: 1=manual, 0=auto) |
| 14 | Delay of selftimer (appears to be time in milliseconds) |
| 16 | Flash mode (s3: 2=flash closed, otherwise 0=auto, 1=ON) |
| 18 | Red eye mode (S3: 0=OFF, 1=ON) |
| 19 | Flash slow sync (S3: 0=OFF, 1=ON) |
| 20 | Flash Sync Curtain (S3: 0= first, 1 = second) |
| 21 | ISO value (S3: 0=auto, 1=ISO-HI, or actual ISO: 80,100,200,400,800) |
| 23 | Image quality (S3 values: 0,1,2 from best to worst) |
| 24 | Image resolution (S3 values: 0,1,2,4,8 for L,M1,M2,S,W) |
| 25, 26 | EV correction (positive or negative, 96 units per stop) |
| 28 | Flash correction (same units as 25,26) |
| 32 | Exp bracket range (Same units as 25/26: e.g. 96 = +/- 1 stop range) |
| 34 | Focus bracket range 2=Smallest, 1=Medium, 0=largest |
| 36 | Bracket mode: 0=NONE, 1 = exposure, 2 = focus |
| 37 | Orientation sensor |
| 39 | Chosen Av (by user) |
| 40 | Chosen Tv (by user) |
| 65 | Focus distance |
| 67 | Focus ok: 1=Yes, 0=NO |
| 68 | Coming Av |
| 69 | Coming Tv |
| 74 | AE lock: 1=ON, 0=OFF |
| 126 | Video FPS (15, 30 or 60. Don't change here!) |
| 127, 128 | Video resolution (S3: 2,1 for 640x480; 1,0 for 320x240) |
| 177 | intervalometer: #of shots (0 if not activated) |
| 205 | 0 '1' during shooting process |
| 206 | "MyColors?" mode (See link below) |
| 218 | Custom timer continuous: # of shots to be taken |
| 219 | Self Timer setting: 0=2 sec, 1=10 sec, 2=custom continuous |

**Special Build USB Remote Routine:**

Put 3v to USB cable to trigger Remote Cable Release

```
@title Remote button
:loop
wait_click 1
is_key k "remote"
if k=1 then shoot
goto "loop"
end
```

**Fingalo Build Alt. Method:**

```
@title Remote button
while 1
  wait_click 1
  if is_key "remote" then shoot
wend
end
```

Note: With this remote function the camera will not enter download mode when connecting the USB cable. For download just disable CHDK by turning of write protect on the SD card.

**Special Fingalo Build uBASIC Syntax**

a=get_vbatt (inserts battery mV into variable a)

set_raw a (a=0 RAW recording off, a=1 RAW recording on)

**Loop commands (do in lowercase, here in UPPER for clarity):**

FOR / TO / STEP /.../ NEXT (step may be + or –)

DO /.../ UNTIL (exit when "until" is true)

WHILE /.../ WEND (loop as long as "while" is true)

IF /.../ THEN /.../ ELSE /.../ ENDIF (multiple relation statements)

## Notes: