

uBASIC User Guide

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

Contents

Preface.....	7
Starting Out.....	8
The Script Header.....	9
The Basics of uBASIC Programming.....	10
uBASIC variables.....	10
Labels.....	10
‘Restore’ label.....	10
Math Expressions allowed in uBASIC.....	11
Logic Operators: AND, OR, NOT.....	12
Sub-Routines.....	12
Logic Commands.....	13
cls.....	13
do/until.....	13
end.....	13
exit_alt.....	13
for/to/next.....	13
get_day_seconds.....	14
get_tick_count.....	14
gosub.....	14
goto.....	15
if/then/else.....	15
let.....	16
print.....	16
print_screen.....	17
random.....	18
rem.....	18
select/case function.....	19
sleep.....	19
while/wend.....	19
Special Build Commands.....	20
Fingalo’s Builds.....	20
for/to/step/next.....	20
if/then/else/endif – multiple ‘if’ statements.....	21
Microfunguy’s SDM (Stereo Data Maker) Builds.....	21
Camera Operation Commands.....	21

autostarted.....	22
click/press/release.....	22
“display”	22
“erase”	23
“iso”, “flash”, “mf”, “macro”, “video”, “timer” (S-series only)	24
“menu”	24
“print”	24
“set”	24
“shoot_full”	25
“shoot_half”	25
“up”, “down”, “left”, “right”	25
“zoom_in”, “zoom_out”	25
get_autostart	25
get_display_mode.....	25
get_drive_mode	26
get_ev	26
get_exp_count.....	26
get_flash_mode.....	26
get_flash_ready.....	26
get_focus.....	26
get_histo_range.....	26
get_IS_mode.....	27
get_iso.....	27
get_mode	27
get_movie_status	27
get_nd_present.....	27
get_orientation_sensor.....	27
get_platform_id	28
get_propset	28
get_quality	28
get_resolution	28
get_shooting	28
get_temperature	28
get_time	28
get_av/tv, set_av/tv, etc	29
get_zoom.....	30

get_zoom_steps.....	31
is_key.....	31
is_pressed “key”.....	31
Motion detection commands.....	31
md_detect_motion.....	32
md_get_cell_diff.....	33
playsound.....	35
press.....	36
release.....	36
set_aflock.....	36
set_autostart.....	36
set_backlight.....	36
set_ev.....	37
set_focus.....	37
set_iso.....	37
set_movie_status.....	37
set_quality.....	37
set_resolution.....	37
set_av/tv, etc.....	37
set_zoom, set_zoom_rel, set_zoom_speed.....	38
shoot.....	38
shot_histo_enable.....	39
wait_click and is_key.....	39
Special Build Commands.....	39
Fingalo’s Builds.....	40
get_prop/set_prop.....	40
S3IS PropIDs.....	40
Digic II PropIDs.....	42
Digic III PropIDs.....	48
is_key – optional method.....	52
set_led.....	52
set_prop.....	53
set_raw.....	53
set_raw_nr.....	54
Allbest’s Builds.....	54
Get ops commands.....	54

get_autostart	54
get_av96	54
get_bv96	55
get_day_second	55
get_disk_size	55
get_dof	55
get_far_limit	55
get_focus	55
get_free_disk_space	55
get_hyp_dist	55
get_iso	55
get_iso_market	55
get_iso_mode	55
get_iso_real	55
get_jpg_count	56
get_near_limit	56
get_prop	56
get_raw_count	56
get_raw_nr	56
get_sv96	56
get_tick_count	56
get_tv96	56
get_usb_power	57
get_user_av_id	58
get_user_av96	58
get_user_tv_id	58
get_user_tv96	59
get_vbatt	59
Set ops commands	59
set_autostart	59
set_av	59
set_av_rel	59
set_av96	59
set_av96_direct	59
set_iso_mode	59
set_iso_real	59

set_nd_filter.....	59
set_prop.....	60
set_raw.....	60
set_raw_nr.....	60
set_sv96.....	60
set_tv.....	60
set_tv_rel.....	60
set_tv96.....	60
set_tv96_direct.....	60
set_user_av_by_id.....	60
set_user_av_by_id_rel.....	60
set_user_av96.....	60
set_user_tv_by_id.....	60
set_user_tv_rel_by_id.....	60
set_user_tv96.....	61
shut_down.....	61
wheel_left.....	61
wheel_right.....	61
Microfunguy's SDM (StereoData Maker) Builds.....	61
USB Remote Cable-Release Function!.....	61
Debugging: the Unk Alert.....	62
Debugging Scripts on a PC or Mac.....	63
Using UBDebug – an Integrated Development Environment for Scripts.....	63
Using the UBASIC_TEST.EXE Console.....	63

Preface

This *uBASIC User Guide* was prepared using the HTML page [UBASIC_TutorialScratchpad](#), with additions from various other pages (especially [PropertyCases](#) and [MoreBest New Scripting Commands](#)) and a few edits based on my own (limited!) experience using CHDK (a470-101b-0.9.8-778) and writing scripts for it.

As you would expect with a community-based development project, those source pages are a mishmash of information from many different authors, sometimes based on different versions of CHDK code, and quoting results from many different camera models.

As a result, it is not possible to prepare a definitive ‘uBASIC User Guide’ that can be reliably used for **all** cameras and **all** versions of CHDK. When features from ‘fringe’ versions of CHDK are found to be useful by many users, they are often then incorporated into the ‘main’ release of CHDK.

So it’s possible that features described in this manual as being “only in so-and-so’s build” might in fact be found in the main release at some stage, if not already. Where I’ve found this to already be the case, I’ve transferred those ‘special’ command instructions into the ‘main’ section, but there could be many others still in the ‘specials’ sections that are now ‘mainstream’ (I don’t have time to test them all!). Rather than trawling the many discussion forums trying to ascertain what’s included and what’s not in your CHDK version, it’s often quicker to simply write a little test script and run it on your camera with your version of CHDK to see whether that feature works!

Draft 4 (D0_4) is a major reorganisation of this document. I’ve added a whole raft of ‘new’ commands, and made an attempt to bring some order to all the included commands, by listing them alphabetically within the ‘type’ groupings (‘logic’ and ‘camera’ commands). Where more than one command was previously listed together (e.g. `get_focus/set_focus`), I’ve separated most of these into their own sections for more easily finding them, but added a ‘see also’ hyperlink to the related command(s).

And the ‘Quick Reference’ section has been dropped. It was a pain to maintain the commands in two locations, and now that this document is bookmarked (and the Table of Contents hyperlinked), you can quickly jump to any command explanation by just keeping the TOC or bookmarks on screen (and quickly get back to the TOC with Adobe Reader’s <Back> button), so the Quick Reference is now redundant.

I doubt that I’ll be spending much time on the various CHDK websites and discussion groups (my life is just too busy for that!), but I’m quite happy to maintain this manual (if anyone other than me finds it useful) if program authors and users are prepared to contact me (preferably e-mail me directly) about corrections/changes/updates.

I haven’t tested most of the Internet links in this document (copied from the original web pages), so please let me know if any are ‘broken’.

Note that where “I...” appears, this is usually a comment from the original author of that section, not me. Where I’ve added a comment, I’ll usually identify myself with ‘DF’.

This manual was prepared using Microsoft Word 2000, and converted to bookmarked/hyperlinked PDF using EasyPDFprinter5. It is formatted for double-sided printing on A4 paper.

Danielf

(dfnojunk at shoalhaven dot net dot au)

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, but even then watch out not to use TAB for indenting (e.g. Notepad or [preferably] Crimson 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).
- You are limited to 8k (8192 bytes) for the size of your script in 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. Try to keep your version of CHDK up-to-date, from the [Wikia website](#).
- **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.
- Keep in mind the scripting and CHDK works by stealing time from a cooperative multitasking system – you're not writing in a time-slicing system like Linux or Windoze. You need to yield back to the camera with a 'sleep...' so the camera can do its stuff reliably as well. For example, some sleep after print gives the camera time to write to the LCD without corrupting what's already on the display. 'Sleep 100' seems to be a commonly-used value, but depending on the camera and other factors, you may need to sleep for 300–400 ms after certain operations. Failure to do this may result in (intermittent!) camera lock-ups or shutdowns (there's a 'watchdog' in the camera turns it off when something goes badly wrong).
- 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 (20 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 *user-definable* variables used in your script. These are often used to set how many exposures you want, how long 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 **z**. 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 (e.g. number of shots, how many steps, etc.). Maximum length of label text is 26 characters (including spaces) (to fit in Parameters Menu space).

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 script's filename 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). For example:

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

If your minimum value for a user parameter variable **a** was 2, then if they changed that setting at 0 (zero) or 1, the above example will automatically increase that variable's value back to your default of 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 uBASIC script writer.

The Basics of uBASIC Programming

'Scripts' are simply small programs running in the CHDK environment, designed to perform some function in your camera.

Most programs are designed to mindlessly repeat some commands. In order to make them work in the proper order, and with the correct number of repeats, the commands are usually contained in some simple recurring loops with counters, or testing for when some condition has been met before it can go on to 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. Some built-in uBASIC commands to simplify these tasks are: for/to/next (counting), do/until (conditional), if/then/else (conditional), and while/wend (conditional).

In this user guide, commands are classified as either 'Logic Commands' (controlling the flow/sequence of the program) or 'Camera Operation Commands' (controlling or getting the status of some camera operation).

In this version of uBASIC, each line can contain only one command. A uBASIC program should always finish with an 'end' command, so the interpreter knows the script has finished.

uBASIC variables

Variables are represented by single letters of the Latin alphabet: a-z and A-Z. All variables are 32-bit signed integers (range -2147483648 to +2147483647). **However, due to a bug in the current main release, values greater than 6 decimal digits cannot be assigned (i.e. 999,999 is the current maximum decimal value) – DF.**

Note: 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.

When used in Boolean equations, a value of zero is considered to be 'false', while a non-zero value is 'true'.

Labels

A label must be the only statement in a line and start with a colon (:). **Maximum string length?? – DF**

'Restore' label

There's one important pre-defined label in uBASIC:

```
:restore
```

uBASIC tries to jump to label "restore" when you fully press the shutter button (which is how you terminate a script prematurely).

The purpose of this label is to allow scriptwriters to code some ‘clean-up’ routine to restore the camera to ‘normal’ settings after their script has forced some other settings. This could be important, for example, if your script uses AF lock, where aborting prematurely may leave the camera in a state where it cannot focus (without cycling power). So by using the **restore** label you could include code to clear the AF lock and turn the display on.

Math Expressions allowed in uBASIC

From 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

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 $s=(\text{some math expression})$ 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 that effectively does this:

```
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 compute like this:

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

and thus print like this:

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](#).

Logic Operators: AND, OR, NOT

not logical **not**. Best to use in a form with parentheses i.e. **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!

Sub-Routines

For complex programming tasks, it is often helpful to split the program into smaller ‘subroutines’ that can be called with the **gosub** command. A subroutine 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 from within the main program by using **gosub “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.

Logic Commands

cls

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 **cls** command.

do/until

A 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
```

end

This should be the last line executed in your script. It tells the script to cease all operations and return camera control back to you. Before **ending** 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.

exit_alt

This command leaves the <Alt> mode.

for/to/next

These are used to set up simple counting loops. For 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. 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 terminal a value has been reached.

For example:

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

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 command.

get_day_seconds

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](#). *(That’s a fairly ‘messy’ example. A better example might be ‘NiteDay2.bas’. – DF)*

get_tick_count

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
```

gosub

Calls the named subroutine, which executes and then returns to the command immediately following the ‘gosub’.

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
end

```

```

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

```

```

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

```

goto

Jumps immediately to the named label.

Below is an example using the **GOTO** command taken from an endless intervalometer script. NOTE: This situation creates an endless loop. Until you manually exit the script (by pressing the SHUTTER button) it will continue. This is generally considered BAD FORM! Any script should 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"

```

if/then/else

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 gosub "subroutine1"
```

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

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

In this case if **a** is NOT greater than the value of 2, your program will call 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)

let

This one is simple. If you see a command that says "let **a** = 2" then that's exactly what happens. It assigns 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.

print

This will print to your LCD or EVF display in the mini-console area (see firmware usage) the text or values that follow the statement, 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 expressions in your **print** statement. [CHR\$() is **not supported** (indeed, it crashes CHDK!), nor is PRINT USING... - DF]

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 "Shot 1 of", a
shoot
for n=2 to a
  sleep t
  print "Shot", 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
```

However, note that a semicolon at the **end** of a print statement, e.g.

```
print "c=", c;
```

does not suppress the newline (CR/LF) like it should! – DF

print_screen

Whatever the script prints on the mini-console screen can also be written to a file on the camera's SD card, such as '\CHDK\LOGS\LOG_0001.TXT'.

To **enable** this console capture, include this initial command in your script:

```
print_screen n
```

where **n** is a non-zero positive integer constant, variable or expression with a value from 1 to 9999. A new file (e.g. for n=1, \CHDK\LOGS\LOG_0001.TXT) is created, or if it already exists, it is emptied (i.e. the new dumped text will **replace** whatever it previously contained).

To turn **off** screen dumping, include this script command:

```
print_screen 0
```

Screen dumping can be turned on and off multiple times in one script, but keep in mind the need to increment the value of 'n' if you don't want to lose the earlier log contents.

The value of 'n' **can** exceed 9999, but the filename will still contain only 4 digits, and will thus simply 'wrap' around. For example, if n=10000, the filename will be LOG_0000.TXT, if n=10001 the filename will be LOG_0001.TXT, and so on. Negative values of 'n' can also be used, but they will be converted to positive by CHDK. So, for example, n=-2 will produce a filename LOG_0002.TXT. In other words, you are limited to a total of only 10,000 'print_screen' log files (should be enough for most people! ☺).

Example:

```
@title printscreen test

@param a None
@default a 0

@param n FileNum
@default n 1

print_screen n
print "START", n
print_screen 0
n=n+1
print "Not written to file"
print_screen n
print "This should be written to next file."
print "a=";a
print_screen 0
end
```

This would create two files with the following contents:

\CHDK\LOGS\LOG_0001.TXT:

```
START 1
```

\CHDK\LOGS\LOG_0002.TXT:

```
This should be written to next file.
a=0
```

random

Returns a random integer between the 'min' and 'max' values.

Example:

```
playsound random 3 6
```

will play the sounds 3, 4, 5, 6 in random order (if repeated in a while loop).

rem

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 when the script is run, however the uBASIC interpreter spends time 'parsing' the rem statement before deciding to ignore it. And 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.

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).

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
```

select/case function

Evaluates a given expression, and then executes one of several commands depending on the value. (For ‘real’ programmers, this is similar to a C ‘switch’ statement ☺. – DF)

<http://chdk.setepontos.com/index.php/topic,1995.0.html>

Syntax:

```
select EXPRESSION
  case EXPR. [to EXPR.] | [,EXPR.2[, ...]]; STATEMENT
  .
  .
  [case_else STATEMENT]
end_select
```

Example:

```
for x=1 to 7
  select x
    case 1; gosub "new"
    case 7; goto "EXIT"
    case 2,4; print "x=2 | x=4"
    case 2 to 5; print "x=3 | x=5"
    case_else print "x=6"
  end_select
next x
:EXIT
print "ready"
end
:new
  print "sub"
  for y=1 to 2
    select y
      case 1; print "y=1"
      case_else print "y=2"
    end_select
  next y
return
```

The second ‘select’ example above (in the ‘new’ subroutine) is in fact a poor example of its use, as this would be better done with a single if/then/else statement!

sleep

This pauses the script to allow some action to take place, or to delay when the next action should occur. It is often necessary to insert sleeps in your script to allow the camera time to execute essential actions – see “Starting Out” for an important note about this.

Syntax: **sleep x**

where **x** is any variable or whole number. The value is in 1000^{ths} of a second (milliseconds, abbreviated ‘ms’), but timer resolution is only around 10–30 ms.

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

while/wend

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.

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 didn'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 guide 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

A standard BASIC **step** command was added to the **for/to/next** commands to 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 constant, variable or math expression, and **step** can be any defined constant, variable or math expression. The step value may also be negative to increment in reverse (i.e. decrement).

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 each time, shoot an image, and exit when the focus has reached or passed 4500mm.

if/then/else/endif – multiple ‘if’ 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.

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.

However this bears little resemblance to uBASIC, and is only mentioned here as a courtesy for those with a special application that might make use of these specialised commands.

See <http://chdk.setepontos.com/index.php/topic,3101.0.html>.

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
```

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 place 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"
```

autostarted

Returns 1 (true) if script was auto-started.

click/press/release

Simulates actuation of a camera button, as described above, the available button names being listed below.

“display”

The **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.

“erase”

The `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 tsteps
@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.

“iso”, “flash”, “mf”, “macro”, “video”, “timer” (S-series only)

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.)

“menu”

The `menu` button.

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

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.)

“print”

The `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.

“set”

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.

“shoot_full”

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

“shoot_half”

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 2 or more 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. **Even better is to follow the press “shoot_half” with:**

```
do
  get_prop 206 p
until p=1
```

[or the equivalent for your camera model] so that it waits only as long as needed for the focus-lock. – DF)

“up”, “down”, “left”, “right”

The “Omni-Selector” (navigation) buttons.

“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 or 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.

get_autostart

Read conf.script_autostart (0=off, 1=on, 2=once).

(See also set_autostart.)

get_display_mode

Returns display mode (record mode only) regardless of Digic II or Digic III camera (one of the features to help get CHDK scripts universal):

- 0 = show info icons
- 1 = do not show info icons
- 2 = LCD off
- 3 = EVF)

get_drive_mode

Returns drive mode (as described in the property pages):

- 0 = single
- 1 = continuous
- 2 = timer (on Digic II)
- 3 = timer (on Digic III)

The timer issue will make it somewhat difficult to use it in universal scripts, however together with a `get_propset` this should be do-able.

get_ev

Returns propcase 25/107 (Digic II/Digic III).

get_exp_count

Will return number of shots in a session (afaik). Useful for counting when you are, for example, in burst mode and want to stop after 3 shots. On SD1000 returns file number of last exposure.

get_flash_mode

Returns flash mode (0, 1, 2 = flash auto, on, off).

get_flash_ready

Returns (*non-zero??*) if flash is ready and charged.

get_focus

(CHDK Build 125 or greater)

Syntax:

`get_focus x` (the distance value [in mm] is placed in variable x)

(See also `set_focus`.)

get_histo_range

Usage: `get_histo_range <from> <to>`

returns percent of values between <from> and <to> in histogram data.

This command gets the information about the histogram of the last image that was shot (i.e. after you have focused, metered, shot etc.), it operates reading the RAW buffer (but doesn't require raw saving to be enabled: the shot histogram data is obtained from the RAW data before discarding or further processing it).

It can be very useful to get information about very dark scenes: the live preview screen is completely black, while the RAW buffer (with a long enough exposure) would contain an image.

- It depends on the `shot_histo_enable` command: if `shot_histo_enable` is not set, `get_histo_range` returns a value of -1, so `shot_histo_enable` has to be enabled first.

This is necessary because the shot-histogram is calculated just after each shot and takes some time. So, by default the shot-histogram is disabled and you need to enable it with `shot_histo_enable`.

-
- Find more about this in this forum thread: [Timelapse with variable shutter speed](#)

Example:

```
rem enable shooting histogram
shot_histo_enable 1
shoot
sleep 100
rem let's read histogram of the shot that was just taken
h=get_histo_range 100 800
rem h contains the percentage of pixels that are
rem between 100/1024 and 800/1024
```

get_IS_mode

Returns IS mode.

0, 1, 2, 3 = continuous, shoot only, panning, off

get_iso

(CHDK Build 125 or greater)

Syntax:

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

(See also set_iso.)

get_mode

(since changeset #497)

Returns whether record mode or playback mode is active. Useful for shutting off scripts that are meant to be run in a certain mode (like 99% of all available scripts!).

Returns **0** in record mode, **1** in playback mode, **2** when mode-dial is set to video-mode AND camera is in record mode (*2 since changeset #499*)

get_movie_status

Returns status of movie recording. Can be used for stuff like checks instead of sleep commands.

- 0 – movie recording is stopped
- 1 – movie recording is paused
- 4 – recording is in progress
- 5 – recording is stopped but camera is still writing movie to SD card

get_nd_present

Returns 0 if no ND (neutral-density filter) present, 1 if ND present and real diaphragm NOT present, 2 if both ND and diaphragm present.

get_orientation_sensor

Regardless of Digic II/Digic III it will return the orientation sensor's degrees.

get_platform_id

(since version 0.8.3/changeset #593)

Returns `platformid` (integer). With the help of this you can code platform-dependent scripts.
→ [Platform-ID list](#)

get_propset

Returns the propset, meaning which set of propsets should work for your camera in regards to Digic II/Digic III. Returns 1 (for propset 1) or 2 (for propset 2). Find your propcases here: <http://chdk.wikia.com/wiki/PropertyCase>

This is one more command that belongs to the “universal category”.

get_quality

(since changeset #497)

Returns the current capturing quality setting in Canon grades, regardless of the cameras OS. For possible values see [PropertyCase](#) (basically quality can be 0–2, with 2 being the worst).

get_resolution

(since changeset #497)

Returns the current capturing resolution setting in Canon grades, regardless of the cameras OS. For possible values see [PropertyCase](#) (basically resolution can be 0–8, corresponding to the Canon settings)

get_shooting

Returns propcase 205 (Digic II)/206 (Digic III).

get_temperature

Usage:

```
get_temperature n
```

where: 0 returns optical

1 returns CCD

2 returns battery temp

This can be used to both assign a variable or to print directly.

Examples:

```
a = get_temperature 0
print "Optical Temp:", a
or:
```

```
print "Battery Temp:", get_temperature 2
```

get_time

(since changeset #497)

Returns elements of the camera’s current time.

Usage:

get_time **0** (returns second)
 1 (minute)
 2 (hour)
 3 (day)
 4 (month)
 5 (year)

get_av/tv, set_av/tv, etc

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). There's a test script for these commands in the "user written scripts".

The commands are:

```
get_av target
get_tv target
set_av val
set_av_rel val
set_tv val
set_tv_rel 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 (this might make bracketing easier and faster):

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

The Av and Tv settings provide the following actual values; roughly ± 1 setting means $\pm 1/3$ EV change (CHDK Build 119 remapped values):

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

Canon firmware uses these ‘?v’ values in many ways, depending on operating mode. The most usual way is that they are set by auto-exposure during half press. Thus to override auto-exposure these prop_cases typically must be set after auto-exposure is finished. Otherwise Canon firmware would just overwrite it and the command wouldn't have any effect.

get_zoom

(CHDK Build 119 or greater.)

Syntax:

`get_zoom x` (zoom-step 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)

(See also `set_zoom`, `set_zoom_rel`, `set_zoom_speed`.)

get_zoom_steps

Returns number of maximum zoom steps, regardless of camera hardware (this will allow for universal zoom scripts). This means, for example, on A620 it will return 8, on S3IS it will return 128.

is_key

(See ‘wait_click and is_key’.)

is_pressed “key”

Checks if a key is being pressed, **when the command is called**. See also ‘wait_click and is_key’ for an after-the-event key-check.

Motion detection commands

Note: 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 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 was 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 COLUMN
  | | | | | | | | | | | | | 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.

playsound

Plays any of the built-in sound events, **if sounds are turned on.**

Example:

```
playsound 0
```

where 0 is the startup sound. There are sounds ranging from 0 to 7, 7 being a nasty long beeeeeeeeeeeeeeeeeep. The first few sounds can only be played if they are NOT muted by the camera, the other beeps will be played though (cam must be un-muted, wasn't tested with the nasty error beep). This feature will lead to many more features, for example anti theft protection together with DataGhost's Disco lights.

0 = startup sound	4 = short beep
1 = shutter sound	5 = af (auto focus) confirmation
2 = button-press sound	6 = error beep
3 = selftimer	7 = long beep (nasty! – can be stopped by ½-pressing the shutter)

press

(See click/press/release.)

release

(See click/press/release.)

set_aflock

(since version 0.9.2/changeset #681)

Lock/Unlock the autofocus.

Usage:

```
set_aflock(x)
```

where **1** locks the autofocus, and **0** unlocks it.

It is like MF, but better in several ways, such as:

- a) available on ALL cameras
- b) focus is locked even after camera returns from deep display sleep (via display key cycle or print button shortcut), which will be VERY good for time-lapses regarding power consumption, camera shake, an actual fixed focus throughout the whole time-lapse and also less stress on the mechanical parts of the camera.

You can use it like this:

```
halfpress (so autofocus is set)
```

```
... (do we need a pause or status check here to wait for focus to be attained? – DF)
```

```
aflock(1)
```

The focus is now FIXED, meaning another half-press will NOT refocus (the AF lamp doesn't even turn on). When you don't need the fixed focus anymore, call **set_aflock(0)**.

Note: AE (Exposure) is NOT affected by this.

set_autostart

Write conf.script_autostart (0=off, 1=on, 2=once).

(See also get_autostart.)

set_backlight

(since version 0.9.1/changeset #679)

With this function the LCD Backlight (or EVF backlight) can be turned on and off – useful for energy-saving or camouflage missions. (Backlight is a bright light inside the LCD, making the image visible.)

Usage:

```
set_backlight x
```

where **0** disables backlight and **1** enables it.

- Note: Camera enables backlight after each shot. To keep backlight mostly disabled, this command must be called after each shot. But do not call it immediately after a shot or its effect will be very short-lived. You must wait for LCD live view image exposure to settle before calling. The regular shoot state loop may help, possibly with some added delay. Test in varying lighting conditions to see if you got it right.

-
- Note: If you accidentally exit a script while backlight is disabled, you can get it back by leaving ALT mode and shooting, or by powering off. The LCD can probably be somewhat read even when backlight is disabled, if bright light is pointed at it from a suitable angle.
 - An example script can be found here: [\[3\]](#)

set_ev

Sets propcases 25 & 26 (Digic II)/107 & 207 (Digic III).

set_focus

(CHDK Build 125 or greater)

Syntax:

set_focus x (where x is the distance in mm)

(The value for 'infinity' is 65535, for some cameras at least [probably all].)

(See also `get_focus`.)

set_iso

(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).

(See also `get_iso`.)

set_movie_status

Is experimental, and so far only works properly on A720 & S3IS. Using this command you can pause, un-pause and stop video recordings.

set_movie_status 1 will pause movie (will not work on all cameras, typically they freeze or shutdown after 2 seconds!)

set_movie_status 2 will un-pause

set_movie_status 3 will stop video recording (this should work on all types of cameras)

set_quality

(since changeset #497)

Set the quality (in Canon grades), usually 0–2. See [PropertyCase](#) for possible values.

set_resolution

(since changeset #497)

Set the resolution (in Canon grades), usually 0–8. See [PropertyCase](#) for possible values.

set_av/tv, etc

(See `get_av/tv`, `set_av/tv`, etc.)

set_zoom, set_zoom_rel, set_zoom_speed

(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 \pm relative change)

set_zoom_speed x (where x can be from 5–100. [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 zooming. Use a click or press/release “shoot_half” command to perform a refocusing, if needed.

Note 2: It was found that if using the slowest speed (5), 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. Five 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 move some of your optics such that 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.

(See also *get_zoom*.)

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/press/release “**shoot_half**” and “**shoot_full**” commands, when used in conjunction with the get_av/tv, set_av/tv, etc and set_user_av_by_id_rel commands.

shot_histo_enable

Setting to 1 enables building of shot histogram data, 0 disables (see also get_histo_range).

wait_click and is_key

Syntax:

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

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:

```
do
  wait_click
  is_key k "set"
until k=1

: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"
end

:k_left
...
return

:k_right
...
return

:k_set
...
end
```

For a special use, see USB Remote Cable-Release Function!

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 didn'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 guide 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](#)

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 commands from MX3 as outlined above. See [MX3's Motion Detection Commands](#)

get_prop/set_prop

Read/set property-case values.

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.

Property cases are different for Digic II and Digic III cameras.

EXAMPLE: A570IS ISO is at #149, not #21 like on the S3IS, and the white balance control is #268, not #206. Furthermore, some Digic III cameras (G7, SD800IS, SD900) use the Digic II property set.

You are advised to confirm that these properties are appropriate for your camera before you attempt to change them.

Usage:

get_prop propid value

set_prop propid value

where **propid** may be as shown in the following tables:

S3IS PropIDs

(see later tables for Digic II & Digic III)

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)

PropID	Description
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)
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 640×480; 1, 0 for 320×240)
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

Digic II PropIDs

(and some Digic III cameras: G7, SD800IS, SD900):

#	R/W	Description	A620	S3IS	SD900
0,236		Shooting MODE Dial position		*	*
1		Photo effect			
2		Custom MyColors Sharpness			
3		Custom MyColors Saturation			
4		Custom MyColors Contrast			
5		White Balance Mode (0, 1, 2, 3, 4, 5, 7 = Auto, Day Light, Cloudy, Tungsten, Fluorescent, Fluorescent H, Custom)			
6		Drive mode (0=normal, 1=continuous, 2=timer)			
8		Hi-speed continuous (1=OFF, 0=ON)			
9		Metering mode (0=eval, 2=centre, 1=spot)			
10		Spot AE Point (0=centre, 1=auto focus point)			
11		Macro mode (0, 1, or 5 for normal, macro, super macro)			
12		Manual Focus (0=Auto, 1=Manual)			
13		AF S3IS: 0=Single, 1=continuous		*	
14		Delay of self-timer (in ms)			
15		Flash adjust mode S3IS: 0=Auto, 1=Manual (see 28 & 29)		*	
16		Flash mode (0, 1, 2 = flash auto, flash on, flash off)			
18		Red-eye mode (0=OFF, 1=ON)			
19		Flash slow sync (0=OFF, 1=ON)			
20		Flash Sync Curtain (0=1 st , 1=2 nd)			
21		ISO value S3IS: 0=auto, 1=ISO-HI, or actual ISO: 80, 100, etc		*	
23		Image quality S3IS: 0, 1, 2 from best to worst		*	
24		Image resolution S3IS: 0, 1, 2, 4, 8 for L, M1, M2, S, W		*	

#	R/W	Description	A620	S3IS	SD900
25,26		EV correction (units: ± 96 per 1 stop range) [On S3is and I guess on more models: Also used in movie mode(!), but AE-lock has to be activated \rightarrow 205 to 1]			
28		Flash correction (units: ± 96 per 1 stop range, if #15=0)			
29		Manual flash output (units 0, 1, 2 from less to full, if #15=1)			
32		Exposure bracket range (units: 96 per 1 stop extension)			
34		Focus bracket range (2=Smallest, 1=Medium, 0=largest)			
36		Bracket mode (0=none, 1=EV, 2=focus)			
37		Orientation sensor SD900: 0=Normal, 270=Left, 90=Right			*
38		Automatic Image Rotation S3IS/SD900: 1=ON, 0=OFF		*	*
39		Chosen Av (User AV)			
40		Chosen Tv (User TV) SD900: 65152=15", 65184=13", 65218=10", 65248=8", 65280=6", 65312=5", 65344=4", 65376=3.2", 65408=2.5", 65440=2", 65472=1.6", 65504=1.3", 0=1"			*
46/47		Seem to be influenced by stitch mode, I guess this is important for Exif data \rightarrow data for stitch programs or the canon stitch assistant – assumption!			
52		Stitch Mode S3IS: 0=from left to right, 1=from right to left, 2=bottom to top, 3=top to bottom, 4=top-left \rightarrow top-right \rightarrow bottom-left \rightarrow bottom-right		*	
57		Digital Zoom S3IS: 0–6, 0 is no digital zoom, 6 is “most digital zoom” (48x); SD900: 3=2.3x, 1=1.4x, 0=no Zoom/Default		*	*
58		Digital Zoom State S3IS: 1=On, 0=Off SD900: 2=1.4x/2.3x, 1=Default, 0=Off		*	*
63		AF-light S3IS/SD900: 1=ON, 2=OFF		*	*
64		Manual settings			
65,66		Focus distance			

#	R/W	Description	A620	S3IS	SD900
67		Focus OK (green, not yellow) when in photo mode 1=OK, 0=not set			
68		Coming Av			
69		Coming Tv			
71		422			
72		Looks like it's \approx ISO * something + 380			
73		371			
74		AE lock activated S3IS: 1=yes, 0=no; see 205 (also in movie mode)		*	
75		0			
76		0			
77		min available Av			
78	R	Flash fired			
79	RW	Flash fire			
80		1			
81		1			
82– 85		Often the same, sometimes slightly different			
86		Av			
87		0			
88		2			
89		0			
90		0			
91		5			
92		0			
93		0			
94		0			
95		8			
96		10			

#	R/W	Description	A620	S3IS	SD900
99		Zoom step S3IS: 0 = open wide, 128 = maximum zoom, on A620 this will be 8 I guess		*	
100		Custom White Balance (28 bytes)			
101		Constantly varying with light			
126		Video FPS SD900: 30=30FPS, 15=15FPS			*
127		Video resolution SD900: 3=1024×768, 2=640×480, 1=320×240, 0=160×120			*
128		Video resolution = 640×480 SD900: 1=640×480, 0=320×240/1024×768/160×120			*
171		Zoom (<i>what? – DF</i>)			
172		Digital zoom (<i>what? – DF</i>)			
177		Intervalometer #of shots			
178		File numbering (Continuous=0, Auto Reset=1)			
181		Display mode (record mode only) (0 = show info icons, 1 = do not show info icons, 2 = LCD off, 3 = EVF)			
184		Slideshow Settings – repeat slideshow (1 = repeat, 0 = do not repeat)			
185		Slideshow Settings – duration (1 = 3 seconds, 2 = 4s, 3 = 5s, 4 = 6s, 5 = 7s, 6 = 8s, 7 = 9s, 8 = 10s, 9 = 15s, 10 = 30s)			
186		Print Settings/DPOF (1 = standard, 2 = overview, 3 = both)			
187		Print Settings/DPOF (1 = date, 0 = no date)			
188		Print Settings/DPOF (1 = filename, 0 = no filename)			
190		In Postcard mode this determines what will be “printed” into the pictures available on cameras which support this A620: 0 = off, 1 = date, 2 = date & time	*		
192		AF Frame/Flexizone X-position			
193	RW	AFL active/activate AF			

#	R/W	Description	A620	S3IS	SD900
194		Used part of sensor when digital zoomed S3IS: no digital zoom = 2816 then descending with increased digital zoom: 2048,1600,1280,1024,832,704 – can be useful!		*	
195		Max resolution?			
196	R	Language Setting: 2=English, 258=German, 514=French, 770=Dutch, 1026=Danish, 1282=Finnish, 1538=Italian, 1794=Norwegian, 2050=Swedish, 2306=Spanish, 2562=simplified Chinese, 2818=Russian, 3074=Portuguese, 3330=Greek, 3586=Polish, 3842=Czech, 4098=Hungarian, 4354=Turkish, 4610=traditional Chinese, 4866=Korean, 5122=Thai, 5378=Arabic, 5634=Japanese) (Confirmed on S3is & A620 and several other cams) NOTE: I just discovered that this setting directly is related to NTSC/PAL setting. These numbers are valid for PAL. If you set to NTSC, all these numbers are decremented by 1, meaning English is 1 and German is 257 in NTSC (instead of 2 and 258).	*	*	*
200		Selected Movie mode			*
204		Related to flash (at least on S3is) – if flash popped down, value is 4, if up value is 3 (in flash ON) and 0 (in Flash AUTO mode) – on A620 when I disable flash, this is also 4 (I guess we can ignore this value on cameras with a “fixed” flash, and use this on cameras with “pop-up flash” to detect if the flash is popped up!)		*	
205		1 during shooting process (this may be an indication that exposure has been determined; it is set shortly after shoot_half, and remains set if AE lock activated. See #74) (in movie mode this is 1 when you activate AE Lock!!!)			
206		Submode of “MyColors?” mode, including Color accent/swap S3IS: 0–11: off, strong, neutral, sepia, black/white, dia, brighter skin tone, darker skin tone, strong blue, strong green, strong red, custom color		*	*
207		Custom MyColors Red			
208		Custom MyColors Blue			
209		Custom MyColors Green			
210		Custom MyColors Skin Tone			

#	R/W	Description	A620	S3IS	SD900
211		Color Accent color			
212		Color Swap from color			
213		Color Swap to color			
218		Custom timer continuous (value: # of shots to be taken)			
219		Self Timer setting S3IS: 0=2 sec, 1=10 sec, 2=custom continuous		*	
223		Mic recording frequency (0 = 11.025 kHz, 1 = 22.050 kHz, 2 = 44.100 kHz)			
226		MODE Dial – looks like it’s the same as PropertyCase 0 (S3IS: tested in scene mode, cycling through “programs”)		*	*
227		Mic Level S3IS: 1–5, low to high		*	
228		Mic Wind Protection S3IS: 1=ON, 0=OFF		*	
229		IS (0,1,2,3 = continuous, shoot only, panning, off)			
230		Converter (0=off, 1 = wide, 2 = tele)			
231		Color Accent tolerance setting			
233		Color Swap/Accent tolerance			
235		Safety Shift (0 = off, 1 = on)			
236		MODE Dial			–
237		47			
254		Focus, auto, 320–725			
257		Color Swap tolerance setting			
260		500			
262		0 in fireworks or movie mode, otherwise 1			
274		411			
298		201329664			
299		–201264128			

* = camera supports PropertyCase (PropertyCase value may be camera specific)

– = camera does not support PropertyCase (at least value does not change)

Digic III PropIDs

(such as SD1000, A470, A570IS and A590IS) and Digic IV based cameras (such as SX10, SX1) not much tested yet...:

#	R/W	Description
3	RO	Auto-Exposure Lock
4		Exposure bracket range (units: 96 per 1 stop extension)
5		AF Assist Beam Setting (0, 1 = Off, On)
6	RO	Focus Mode (0, 1, 3, 4, 5 = Normal, Macro, Infinity, Manual, Super Macro [SX10])
8		AiAF Mode (0, 1, 2 = On, Off, Face-Detect)
11	RW	Auto-Focus Lock
12		AF Mode (0=single, 1=continuous) [SX10]
18	RW	Auto-Focus state after half-press (stays 1 if shoot_half is released!) (0 orange, >=1 green) <i>(with AiAF=off only 0 or 1, with enabled AiAF or Face the value depends on position and number of AF boxes)</i>
21		Auto Rotate (0, 1 = Off, On)
23	RW	Av (for next shot, updates on half-shoot), 96 per 1 stop
24		Min Av (widest available aperture for current zoom, updates on half shoot), 96 per 1 stop. Note: on SD990, this includes the ND filter, if the ND was selected by the camera in an auto mode.
25		Min Av (widest available aperture for current zoom, updates on half shoot), 96 per 1 stop. Note: on SD990, this does not include the ND filter.
26	RW	User Av (user selected “market” value in M/Av modes; writes become effective on LCD on half shoot), 96 per 1 stop.
29		Bracket mode (0=none, 1=EV, 2=focus)
34		Bv
49,50		MODE Dial
55		Custom Color: Saturation (-2, -1, 0, 1, 2 = 254, 255, 0, 1, 2)
57		Picture Quality (0, 1, 2 = Superfine, Fine, Normal)
59		Custom Color: Contrast (-2, -1, 0, 1, 2 = 254, 255, 0, 1, 2)
60		Converter (0, 1, 2, 4)

#	R/W	Description
61		Language + video standard settings; Value=PAL; for NTSC=value-1 (A720) English=1, German=257, French=513, Dutch=769, Danish=1025, Finnish=1281, Italian=1537, Norwegian=1793, Ukrainian=2049, Swedish=2305, Spanish=2561, Chinese simplified=2817, Russian=3073, Portuguese=3329, Greek=3585, Polish=3841, Czech=4097, Hungarian=4353, Turkish=4609, Chinese traditional=4865, Korean=5121, Thai=5377, Arabic=5633, Romanian=5889, Japanese=6145
63		Number of continuous shots taken last time
64	RW	Flash Sync Curtain (0=1 st , 1=2 nd)
65		Subject Dist 2 (“Near limit”, most linear to real distance. You can get-and-set this prop_id, but it doesn’t affect the near limit and/or the focus – tested with A590).
66	RW	Date stamp (0=Off, 1=Date, 2=Date & Time); only postcard mode
79		Delta Sv (<i>What is this? Someone please add a description.</i>)
91	RO	Digital Tele-Converter mode (>0 if enabled)
92	RO	Digital zoom source image width in pixels
94		Digital zoom mode (1=“Standard”, 0=any other mode)
95	RO	Digital Zoom depth (0–6: 0=none, 1–6 are for camera dependent zoom steps, higher number = higher magnification)
102	RO	Drive Mode (0, 1, 3 = single, continuous, timers)
103		Changes with light only in M-Mode (–445 is completely dark and 328 is fully bright. Also depends on the current av/tv setting.)
105		Display mode (record mode only) (0, 1, 2, 3 = show info icons, do not show info icons, LCD off, EVF)
107, 207	RO, RW	Exposure Shift/Compensation (32 for 1/3 stop)
111		External flash state (0=Absent, 1=Present and turned on, 2=Present but turned off)
113		Focus bracket range (2=Smallest, 1=Medium, 0=largest)
115	RO	Ready to shoot, focus okay.
117		File numbering (Continuous=0, Auto Reset=1)
121	RW	Flash adjust mode (0=Auto, 1=Manual) (see 127&141)

#	R/W	Description
122	RO	Flash used in last shot (0, 1 = Flash not used, Flash used)
127	RW	Flash exposure compensation (units: ± 96 per 1 stop range, if #121=0)
133	RO	Manual Focus Mode (0, 1 = Off, On)
141	RW	Manual flash output (units 0, 1, 2 from min to full, if #121=1)
143	RW	Flash Mode (0, 1, 2 = flash auto, flash on, flash off)
145		IS (0, 1, 2, 3 = continuous, shoot only, panning, off)
149	RW	ISO Mode (0=auto, 1=high auto, 100=manual ISO 100 etc)
155		Metering Method (0, 1, 2 = Evaluative, Spot, Centre)
165	RW	Time lapse movie shoot interval (milliseconds)
166		Movie frame rate
169		Video size mode (160=0, 320=1, 640=2)
170		Video play mode (LP=0, SP=1)
184		My Colors (0–11): 0=Off, 1=Vivid, 2=Neutral, 3=B/W, 4=Sepia, 5=Positive Film, 6=Lighter Skin Tone, 7=Darker Skin Tone, 8=Vivid Red, 9=Vivid Green, 10=Vivid Blue, 11=Custom Color)
195		Optical Zoom position (0, 1, 2, 3, 4, 5, 6...)
196		(? Scene dependent/ stays 0 when scene is black or when scene has optimal light)
206		Shooting in progress
208		Flash Ready (1 during shutter half press after metering if camera is going to use flash)
212		Review info (0=Off, 2=detailed, 3=focus check) [SX10]
213		Red-Eye (0 “Off”, 1 “On”)
218	RW	Picture Size (0, 1, 2, 3, 4, 5, 6, 8 = L, M1, M2, M3, S, RAW (on g9), Postcard, W)
219	RO	Orientation Sensor (0, 90, 270)
220		Safety FE (0 “Off”, 1 “On”)
223		Timer Mode (0, 1, 2 = 2 Second, 10 Second, Custom)
224		Timer Delay (ms) (can’t override)

#	R/W	Description
225		Custom Color: Sharpness (-2, -1, 0, 1, 2 = 254, 255, 0, 1, 2)
227		“Long time exposure” indicator: 1 always in “Night snapshot” scene mode and when the exposure time is set to ≥ 1 s
233		Orientation stitch assistant (left to right=0, right to left=1)
245		Subject Dist 1 (equals ID 65 on the A590)
246		Sv Market (for next shot, updates on half-shoot), 96 per 1 stop
247		Sv (for next shot, updates on half-shoot), 96 per 1 stop
249		Digital Zoom position, same as #95, 0–6 on A590
251		Optical Zoom position, same as #192, 0–128 on SX10, same as #195 on A590, 0–7 on A590
252		Manual focus: equal value like 65, 245
254		Focus related, -1 for infinity
262	RW	Tv (for next shot, updates on half-shoot), 96 per 1 stop
264	RW	User Tv (user selected value in M/Tv modes, writes become effective on LCD on half shoot), 96 per 1 stop
268		White Balance Mode (0, 1, 2, 3, 4, 5, 6, 7 = Auto, Day Light, Cloudy, Tungsten, Fluorescent, Fluorescent H, Flash [SX10], Custom)
269		White Balance Value (Middle around 900. Less corrects red and higher corrects blue. Can’t override.)
277		Safety MF (0 “Off”, 1 “On”)
280		RAW+JPG (Raw+Jpg off = 0, Raw+Jpg on = 1)
290		iContrast setting (0=off, 1=on) [SX10]
293		Servo AF (0=off, 1=on) [SX10]
294		Aspect/Widescreen (1=widescreen, 0=standard) [SX1] ([1])
296	RO	ND filter status (0 off, 1 on) for manual mode. NOT updated in auto modes (SD990). Note: you can write to it, but actual ND state doesn’t change.
297		??? (SX10, SD990: 1 after half-press)

On the A590, it seems that possible propIDs range from 0 to 287. IDs greater than 287 will always return “14”.

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
```

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)
```

set_led

LED lamp control.

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 as 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 is 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.

set_prop

(See get_prop/set_prop.)

set_raw

(Also in AllBest builds)

Enable/disable RAW Recording.

Usage:

```
set_raw a
```

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

set_raw_nr

SET dark frame subtraction state (ON|OFF|AUTO).

(Currently only in Fingalo's v106 or later)

Usage:

set_raw_nr a

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

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.

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](#), namely those that 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_autostart

Parameter checking autostart for scripts.

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

get_av96

Note: Since Canon appears to use integers when calculating APEX values, they multiply what would otherwise be fractional values by 96 (found by analysing their values to match the APEX formula) to get integers with reasonable granularity. As uBASIC is also integer-based,

this is very convenient because we can read and set appropriate values using the propcase `get_prop/set_prop` functions, or specific `get/set` functions that operate on the propcases. – *DF*

get bv96

Get brightness value. (See note at `get_av96`.)

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 disk size

(See `get_free_disk_space`.)

get dof

Get the depth of sharpness in mm.

get far limit

Get the border zone ranged acceptable sharpness mm.

get focus

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 hyp dist

Get hyperfocal distance.

get iso

Obtain ISO mode.

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 jpg count

(See `get_raw_count`.)

get near limit

Get dipped border zone acceptable sharpness.

get prop

Obtain property case, call

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.

get raw nr

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

Syntax: `x=get_raw_nr`

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. (See note at `get_av96`.)

get tick count

Returns system time, in milliseconds since camera power-on.

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

Returns `tv * 96`. (See note at `get_av96`.)

The following table is for DIGIC II and III cameras:

speed	Tv96-val.	speed	Tv96-val.	speed	Tv96-val.	speed	Tv96-val.
64.0"	-576	3.2"	-160	1/6"	256	1/125"	672
50.8"	-544	2.5"	-128	1/8"	288	1/160"	704
40.3"	-512	2.0"	-96	1/10"	320	1/200"	736
32.0"	-480	1.6"	-64	1/13"	352	1/250"	768
25.4"	-448	1.3"	-32	1/15"	384	1/320"	800
20.0"	-416	1.0"	0	1/20"	416	1/400"	832
16.0"	-384	0.8"	32	1/25"	448	1/500"	864
12.7"	-352	0.6"	64	1/30"	480	1/640"	896
10.0"	-320	0.5"	96	1/40"	512	1/800"	928
8.0"	-288	0.4"	128	1/50"	544	1/1000"	960
6.3"	-256	0.3"	160	1/60"	576	1/1250"	992
5.0"	-224	1/4"	192	1/80"	608	1/1600"	1021
4.0"	-192	1/5"	224	1/100"	640	1/2000"	1053

So for DIGIC II and III cameras you can calculate shutter speed from this equation:

$Time = 2^{-\left(\frac{tv96}{96}\right)}$ seconds, though unfortunately CHDK's uBASIC doesn't facilitate this calculation, lacking the '^' (exponentiation) operator ('would be easy in assembler or C ☺).

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_usb_power

Checks for USB power connected. Works for series A and S-as a minimum.

Syntax: `x=get_usb_power`

For G-series it is not working. Integration with USB button. (?)

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 $\frac{1}{3}$ EV)

get user av96

Returns custom av * 96. (See note at `get_av96`.)

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 the 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. (See note at get_av96.)

get vbatt

The voltage of the battery.

Syntax: `x=get_vbatt`

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

get_vbatt acts as its own variable. This allows you to use it even within expressions, e.g. `if (get_vbatt <= 2800) then print "DEAD BATTERY!"`

Set ops commands

(Usually associated with suitable parameters):

set autostart

Use 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.

set av

See set_user_av_by_id (compatibility)

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. (See note at get_av96.)

set av96 direct

Direct installation av * 96. It works similarly to direct the installation of the interface CHDK av. In any mode. (See note at get_av96.)

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 nd filter

Usage:

```
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.

set prop

Install PropertyCase (?).

set raw

The installation script in raw mode; disables the last (previous?).

set raw nr

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

set sv96

Direct installation of the sensitivity of APEX (Sv * 96). (See note at get_av96.)

set tv

See set_user_tv_by_id (interoperability)

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. (See note at get_av96.)

set tv96 direct

Direct tv * 96 installation. Works by installing excerpts from the CHDK interface. (See note at get_av96.) See Example at get_tv96.

set user av by id

Custom install av according to the ID in CHDK.

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 av96

Custom install av * 96 in the manual modes. (See note at get_av96.)

set user tv by id

The installation of custom tv permissible, in accordance with Canon ID for CHDK.

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 tv96

The installation of custom Tv * 96. (See note at get_av96.)

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.

wheel left

wheel right

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 A710IS 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 of the cameras are supported for the Remote USB feature, and most 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:

is_key x “remote”

(Special “**remote**” camera button for use in **is_key** commands.)

Running this small script (or the loop embedded as a subroutine in lengthier 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](#).

Note however, that UBDebug contains the same bugs as the uBASIC interpreter in CHDK, so if your code activates an interpreter bug (as opposed to a bug in your script), the debugger will crash just like CHDK, without giving you any help! – DF

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. (To minimise typing, you might want to rename this to just 'test.exe'.) 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 [better still] by installing rjhExtensions, or with a simple registry hack.) 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 a directory on C:\ and your scripts are on drive D: in a sub-folder called CHDK\Files\Scripts\, at the command prompt just type:

```
d:<Enter>
cd \CHDK\Files\Scripts<Enter>
```

and you'll be where you're supposed to be. (You might want to enter those commands in a file named [say] CDtest.bat to save typing them each time. Make sure the batch file is in your DOS path!)

To test one of your scripts in that folder, at the Command Prompt, just type:

```
uBASIC_test scriptname.bas<Enter>
```

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 that 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_test.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” "%1”
```

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.

Notes: