The Canon Camera Hackers Manual

Berthold Daum

Teach Your Camera New Tricks

rockynook

The Canon Camera Hackers Manual

Berthold Daum

The Canon Camera Hackers Manual

Teach Your Camera New Tricks

rockynook

Berthold Daum, berthold.daum@bdaum.de

Editor: Gerhard Rossbach
Production Editor: Jimi DeRouen
Copyeditor: Cynthia Anderson
Layout and type: Petra Strauch, just-in-print@gmx.de
Cover design: Helmut Kraus, www.exclam.de
Printer: Malloy, Ann Arbor, Michigan
Printed in USA

ISBN 978-1-933952-58-1

1st Edition

Rocky Nook, Inc.
26 West Mission Street, Ste 3
Santa Barbara, CA 93101-2432

www.rockynook.com

Library of Congress Cataloging-in-Publication Data

Daum, Berthold, 1949-
The Canon camera hackers manual / Berthold Daum.
p. cm.
Includes bibliographical references.
ISBN 978-1-933952-58-1 (alk. paper)
1. Canon digital cameras--Automatic control. 2. Digital cameras--Modification. I. Title.
TR263.C3D37 2010
771.3'3--dc22
2010005926

Distributed by O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472

All photographs and illustrations by the author.

This book is printed on acid-free paper.

Table of Contents

1 Introduction

On Sept. 2, 2009, a group of MIT students launched a balloon with a digital camera into near space to take photographs of the earth (http://space.1337arts.com). Five hours later, the camera returned from the flight, having reached a height of 17.5 miles. All of the equipment was put together from off-the-shelf components (balloon, parachute, camera, GPS tracker, etc.) and did not cost more than $150. Several missions similar to this one had been completed before by other groups, but not at such a low cost. For example, in 2008, an Italian group launched a digital camera into space taking both photos and videos (http://www.francescobonomi.it/ICBNN).

Both projects used Canon compact cameras for taking pictures. The MIT students used a *Canon A470*, the Italian group a *Digital Elph SD 1100 IS (Ixus 80 IS)*. In addition, both cameras were equipped with the CHDK, the *Canon Hack Development Kit*. Despite its intimidating name, the CHDK is a ready-to-run piece of software that can be installed on the camera's memory card and that enhances the camera with many new features. In the above-mentioned space missions, the CHDK determined when to begin shooting and in what intervals. In the case of the SD 1100 IS, the CHDK also ran a shutter priority exposure program that the camera normally does not offer.

These are only a few tricks that the CHDK has up its sleeve. RAW shooting, live histograms, remote control, ultrashort and ultralong exposure times, time-lapsing, motion detection, exposure and focus bracketing, camera automation with scripts, and, and, and... More than a few high-priced DSLRs would be proud to have such features. No wonder the CHDK has hit the columns of nearly every photography-related magazine and e-zine around the world.

Today, the CHDK can be regarded as a mature product that is robust enough for the most demanding missions. It is available in many languages. The documentation, too, such as manuals and tutorials, is rapidly improving. There is a lively online community at http://chdk.setepontos.com/.

How this book is organized

We start with an introduction to Canon compact cameras and the operating systems they use. An introduction to the CHDK—what it is and why you should use it—completes chapter 2.

Chapter 3 deals with how to install the CHDK—both utility-augmented installation on a PC and manual installation by utilizing camera functions.

Chapter 4 discusses the out-of-the-box functions of the CHDK. First, we examine how to tame the beast—at times, the information overflow can be overwhelming. Then we unleash it again: advanced exposure control, ultrashort and ultralong exposure times, manual focusing, shooting RAW, bracketing, edge overlay, remote control, etc.

Chapter 5 then turns your camera into a robot via scripting in *uBasic* and *Lua*. After short introductions to these programming languages, we discuss scripts for time lapse, bracketing, motion detection, exposure control, and configuration switching.

Chapter 6 covers some advanced techniques: large panoramas, super-wide-angle shots, and combining *High Dynamic Range (HDR)* photography with panoramas and time-lapse movies.

A CHDK spin-off, the *Stereo Data Maker* (SDM) is discussed in chapter 7. Shooting 3D stereo images is an important topic in this chapter, along with the strong capabilities of the SDM for remote photography, communication with external devices, and Digiscoping.

Application of the SDM (and CHDK) to areas such as *Kite Aerial Photography* (KAP), balloon-based photography, and other remote platforms is discussed in chapter 8.

Chapter 9 takes a short look at similar "camera enhancement" projects, such as the EOS-CHDK, *MagicLantern* (which turns the Canon EOS 5D into a professional movie camera), and the Pentax project.

Finally, the appendix contains information on using memory cards of more than 4 GB with the CHDK. There is a section with important web links and another section with tips in case you run into trouble. Finally, you'll find out how to contribute to the further development of the CHDK.

Acknowledgments

First and foremost, I wish to thank the publishers. Thanks go to my lector, *René Schönfeld,* at *dpunkt.verlag* Heidelberg for co-parenting the idea of writing a book about the CHDK; and to *Gerhard Rossbach,* publisher at *Rocky Nook,* Santa Barbara, for his enthusiasm. Copyediting was done by Cynthia Anderson, typesetting and graphic design by Petra Strauch, Just in Print. Thanks go also to *Alice Philipp,* my partner in life, for her patience and support.

This book would not have been possible without the many volunteers who have contributed to the CHDK in various forms: the initial creators of the CHDK core, *VitalyB* and *GrAnd;* later contributors such as *Fingalo, Juciphox, MicroFunguy,* and others; the volunteers who ported (and continue to port) the CHDK to new cameras; the many script, documentation, and utility authors; and the whole vibrant CHDK community that continues to improve an already strong product. With this book, I hope to give something back to the community.

2 Cameras and Operating Systems

Before turning to the virtues of the CHDK, let's first have a look at some of the platforms the CHDK runs on. As its name suggests, the CHDK runs on *Canon* cameras. Specifically, the classic form of the CHDK runs on Canon compact cameras, such as *Powershot* and Digital *Elph (Digital IXUS)*. For the still-experimental (at time of writing) EOS-CHDK, please see section 9.1.

2.1 Camera hardware

Practically all digital Canon compact cameras (and all of the newer camera models) are equipped with a zoom lens, allowing for an optical zoom range from 1:3 to 1:20+. The optical zoom can be extended via an optional digital zoom. All zoom lenses are equipped with a macro function, which again can be extended with a digital macro. Digital zoom and digital macro come at a cost, however: they reduce image resolution because only a part of the sensor is used for the final image.

Most Canon cameras are equipped with an optical image stabilizer; usually, these cameras carry the suffix "IS" in their name. "Optical" means that a lens element is moved to counteract camera shake. With this technique, it is possible to increase exposure times by a factor of 2–16 (1–4 f-stops) when shooting without a tripod. An image stabilizer is a very useful feature, especially for telephoto shots and for shooting in low light conditions.

Most Canon cameras use CCD (charge coupled device) type sensors; only a few use CMOS type sensors. Smaller models have sensors with a diameter of 1/2.5 inch (10 mm), while larger models have sensors with a diameter of 1/1.7 inch (14.9 mm). The focal length conversion factors are 6.2 and 4.6 respectively, compared to the 35mm format.

For most Canon cameras, the resulting image is delivered in the form of a JPEG file with selectable resolutions and formats, and selectable compression ratios. Top-range cameras offer an option to deliver the RAW sensor data, too. Actually, the prime motivation behind the development of the CHDK was to enable smaller, low-cost cameras to deliver sensor data in RAW format. It should be mentioned that digital zoom and digital macro do not work with RAW format; the format delivers the pure sensor data without any post-processing.

For the smaller models—especially the *Digital Elph (Ixus)* series—the shutter mechanism consists of a simple mechanical shutter plus an electronic shutter. Because the circuitry for the electronic shutter requires some space on the sensor chip, less sensor area is left for the light-sensitive parts—resulting in a less than optimal signal-to-noise ratio compared to a DSLR. Therefore, the top-of-the-range Powershot models, such as the G-series, use a mechanical shutter in order not to sacrifice valuable sensor real estate.

Similarly, the lenses of the low-cost cameras are not equipped with a diaphragm but rather with an ND (neutral density) filter that is switched on when the scene becomes too bright. Because the lens is always used at full aperture, it is not possible to increase the depth of sharpness (DOF) by stopping down with these cameras. Variations in brightness and shutter speed are compensated for by the ND filter and by changing the sensor's sensitivity (ISO speed). Higher-priced cameras are equipped with a traditional diaphragm.

All Canon cameras are, of course, equipped with autofocus (AF) and automatic exposure (AE) control. Newer models even have face detection. Only the top-range models allow for manual control of focus, shutter speed, aperture, and ISO. With the CHDK, manual control of focus and exposure is achieved for all models—within the physical limits of the camera, of course.

All Canon cameras are equipped with an internal flash unit. Some larger cameras offer the option of connecting an external flash as well. Unfortunately, not all cameras allow fine-tuned manual control of the flash power. As a result, flash images can often look unnatural and flat. In many cases, the flash is too bright. The CHDK allows a manual flash on all models with three different power settings, resulting in more natural-looking flash images.

All Canon cameras are equipped with a video function, in some cases an HD video function. Because the camera's image sensor has a much higher resolution than a video frame, digital zoom *(Digital Converter)* can be utilized to a certain degree for video shots without losing quality. In most cases, the optical zoom is disabled during video shooting, as the zoom motor's noise would ruin the soundtrack. With the CHDK, this behavior can be overridden (for some cameras) and the optical zoom can be used while shooting video (section 4.8). Some editing will be required, however, for the soundtrack.

The built-in audio facility (microphone and audio digitizer) that is used for the video soundtrack can be used for photos, too. Voice notes can be attached to photos, or the camera can be utilized as a dictating machine.

2.2 Processors and operating systems

Since 2002, Canon has used the DIGIC processor in all of its digital cameras, including the EOS DSLRs. The original DIGIC processor was followed in 2004 by the DIGIC II processor, which combines all functions on a single chip (the original processor needed three chips). DIGIC II was followed in 2007 by DIGIC III, which brought new features such as face recognition and iSAPS scene-recognition technology. DIGIC IV was introduced in 2008 and features faster and better image processing and live face detection.

DIGIC II and some DIGIC III processors are equipped with the *VxWorks* operating system from *Wind River Systems*. In 2007, Canon introduced its own operating system called *DryOS* that has been used since then on most DIGIC III and DIGIC IV platforms. DryOS can run on more than 10 different processor types.

Compact cameras using the DryOS are: A470, A480, A580, A590 IS, A650 IS, A720 IS, A1000 IS, A1100 IS, A2000 IS, A2100 IS, D10, E1, G9, G10, G11, SD1100 IS (IXUS80 IS), SD770 IS (IXUS85 IS), SD780 IS (IXUS100IS), SD790 IS (IXUS90 IS), SD1200 IS (IXUS95 IS), SD960 IS (IXUS110 IS), SD940 IS (IXUS120 IS), SD980 IS (IXUS200 IS), SD870 IS (IXUS860 IS), SD880 IS (IXUS870 IS), SD950 IS (IXUS960 IS), S5 IS, S90, SX1 IS, SX10 IS, SX20 IS, SX100 IS, SX110 IS, and SX200 IS.

The distinction between the operating systems is important because some CHDK scripting functions depend on the operating systems. Scripts utilizing these functions usually run only on the platform they were developed for, or they must implement special provisions for cross-platform operation. Developing platform-dependent scripts is fine for home use. When publishing scripts for a larger audience, however, portable scripts are definitely the preferred solution. Newer versions of the CHDK allow access to most camera functions in a platform-independent way, so that with some care, portable scripts can be developed without much difficulty.

Figure 2-1
The CHDK logo shows up during the boot process of the CHDK. The logo display can be switched off via the CHDK menu function **ALT** > **MENU** > *Miscellaneous Stuff* > *Show Splash Screen on Load*.

2.3 The CHDK: What it is and how it works

2.3.1 History

The development of the CHDK began in 2006 with the Russian programmer *VitalyB*. Studying the disassembly of a firmware upgrade for a Canon IXUS camera, he was able to analyze the upgrade process and write a program that would boot from the card and take control of the camera. With the help of this program, he was able to read out (dump) the original Canon operating system from the camera to a PC. Data transfer used the blue camera LED and a phototransistor, a method still employed today with some cameras.

The first build of the CHDK enabled the camera (an A610) to deliver RAW 10-bit sensor data in combination with JPEG images, allowing true RAW processing for a consumer camera. Apart from the RAW format, this first version supported live histograms, scripting, and three-exposure bracketing. Scripting was made possible through the inclusion of the *uBasic* interpreter, created earlier by Swedish programmer *Adam Dunkel*.

The program proved so useful that versions for other cameras soon followed. This was first done for cameras with the *VxWorks* operating system; cameras with the *DryOS* operating system should follow later. For each camera (in fact, for each firmware version), it is necessary to read out the firmware from the camera and analyze it in order to link the additional functions to the original firmware (section A.4).

In 2007, another programmer, *GrAnd,* joined the CHDK community and added numerous features such as the on-screen display (OSD) of focal distance, zoom step and factor, hyperfocal distance, DOF calculator, shadow and highlight clipping warnings, battery meter, and much more. An article in *DP Review* about the new features brought public attention to the CHDK.

More developers joined the CHDK community. This initially resulted in different branches of the CHDK, each with its own set of special functions. Particularly notable are *MX3's* build with motion detection, *Fingalo's* build with much-improved *uBasic* scripting functionality and USB support, and *Juciphox's* build with support for *Lua* scripting. Today, the official CHDK—the *MoreBest* build—integrates many of these additional functions.

2.3.2 How it works

Canon cameras perform several steps when switched on:

- First, the camera's firmware checks the memory card. If the card is write-protected, and if it contains a file named `DISKBOOT.BIN` in its root directory, then this file is loaded and executed. This allows almost any program in the camera to be infiltrated.
- If such a file is not found, the normal start-up process continues. This includes the launch of four different native tasks that run in parallel and manage the different units of the camera:
  - The *Logging task* takes care of the camera's display unit. It is responsible for presenting information to the user.
  - The *Keyboard task* monitors the camera's buttons for key presses.
  - The *Image-capturing task* is responsible for exposure and focus control and for reading the data from the sensor.
  - The *File-system task* is responsible for writing images to the memory card and managing the card's file system.

The CHDK adds an additional task that manages the CHDK functions. In addition, the CHDK needs hooks in the native tasks to communicate with them. Therefore, the CHDK loader embodied in the file `DISKBOOT.BIN` performs the following steps:

- Copies the core CHDK code into a suitable memory location within the camera. Of course, this is only possible if the camera has enough free memory. On some cameras this is not the case, and limited or no CHDK support is possible.
- Restarts the camera with the CHDK code as the entry point. Instead of the native boot process, the CHDK boot process is now performed. This process will:
  - Launch the CHDK task.
  - Add hooks to the native tasks *(logging task, keyboard task, image-capturing task, file-system task)* while loading and launching those tasks.

In this way, the changes applied to the native firmware are only temporary and remain minimal. Unfortunately, the position of the hooks is different for every firmware version, so a different CHDK version is required for each firmware version. The main task of migrating the CHDK code to a new camera is to find out the correct position of the required hooks (section A.4).

2.3.3 What the CHDK can do for you

RAW images

One of the most outstanding functions of the CHDK (and historically the first) is the ability to deliver and process RAW sensor data. The popularity of RAW image processing relies on the fact that **RAW files** contain the full information gathered by the sensor. The JPEG format, in contrast, resamples the 10-, 12-, or 14-bit pixel values down to an 8-bit scheme. This can result in washed-out highlights. PC-based RAW converters, on the other hand, are able to recover at least some of the blown-out highlights.

An additional problem with the JPEG format is interpolation. Sensor cells responsive for red, green, and blue light are placed side by side. Therefore, interpolation is required to compute a complete pixel value as required by the JPEG format. PC-based RAW converters (section 4.5.4) can do a better job than the tiny processor within the camera, resulting in improved sharpness and fewer artifacts such as moirés.

Another advantage of the RAW format is that processing decisions, such as white point and color space (e.g., sRGB, AdobeRGB), can be postponed until the image is in the development phase on the PC. There, the photographer can try out different settings without risking loss of image detail.

Figure 2-2
The CHDK main menu organizes the numerous functions into 11 groups. The menu is obtained by changing into *<Alt>* mode (chapter 4), then pressing the **MENU** button.

Recent versions of the CHDK are able to deliver RAW data in the popular **DNG format**, created by Adobe as a camera-manufacturer-independent RAW format. It's a sensible choice; manufacturers of premium cameras such as *Hasselblad, Leica,* and *Pentax* have chosen it as their native RAW output format.

Scripting and motion detection

An outstanding feature of the CHDK is its ability to automate tasks by writing a script. While older versions of the CHDK support only the easy but rather simplistic scripting language *uBasic,* newer versions come with built-in support for the more advanced *Lua* language, which is also used by *Adobe Lightroom* for automation purposes. With scripting, it becomes possible to shoot long time series or to let the camera operate autonomously, as in unmanned aerial missions.

A rather surprising feature of both uBasic and Lua is the possibility of motion detection. Under the control of a script, the camera shoots when motion occurs within the image field, or if a detected motion stops. The CHDK uses the camera sensor for this purpose, allowing highly configurable motion detection without the need for extra hardware.

More control

Most Canon compact cameras operate automatically. If they offer a manual mode, it enables you to dial in correction values for exposure but does not allow you to set explicit values for **shutter speed, aperture,** or **ISO speed,** or to **focus** manually. The CHDK, in contrast, enables you to do exactly that. Not only is it possible to dial in precise values for shutter speed, aperture, ISO speed, or focus distance, but the range of available shutter speeds is significantly expanded. Extremely short shutter speeds, such as 1/60,000 sec., and extremely long exposure times are possible under the control of the CHDK, depending on the physical limits of the camera. For video clips, it becomes possible to influence the **video quality,** and on some cameras, to enable the optical zoom while shooting video.

Better information

While these possibilities extend the application range of the camera, the additional OSD *(On Screen Display)* provides greater control over the general image-making process. This starts with a **battery meter** that gives you a precise readout (in percent) of the remaining battery power. This feature alone makes it worth installing the CHDK, because shooting sessions and battery usage can be planned in a much better way. Precise exposure control is achieved through a highly configurable **live histogram** and the

display of **zebra areas** (over- and underexposed image areas masked with a zebra pattern) that allow you to judge exposure *before* shooting. These information items can be arranged freely on the screen with the help of the built-in OSD editor.

The **Edge Overlay** feature is also very useful. When switched on, the contours of a shot are extracted and appear as an overlay on the display. This allows you to align the following shots to the overlay, and thus to the previous shot. This is particularly useful when shooting panoramas, for bracketing work such as HDR photography or focus stacking, or for shooting 3D stereo images with a single camera. Unfortunately, this feature does not work in video mode where it could be useful for connecting different takes.

Remote control

While Canon EOS cameras allow full remote control, there are no provisions of that kind for most compact cameras. Again, the CHDK adds that feature to supported compact cameras and enables **remote control** through the camera's USB port—including shutter release, zooming, and the possibility of remote functionality in scripts.

Utilities and games

Finally, the CHDK comes with a number of useful utilities such as a **file browser,** which allows you to browse the memory card's file system and to delete files and folders. There is also a **calendar** and a **text file reader** (for example, to read script documentation), and finally, a number of games in case you get bored.

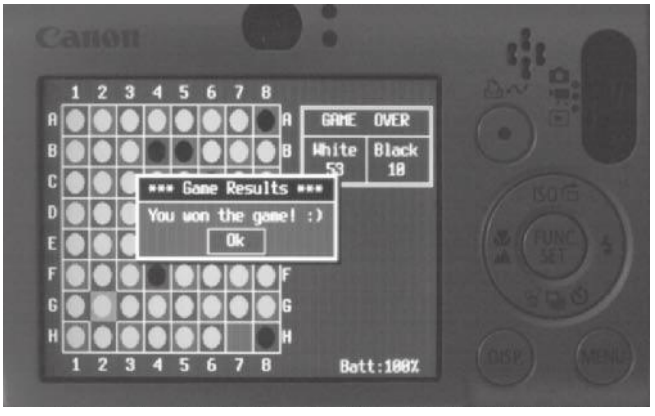


Figure 2-3
An incentive to always take your camera with you: games in the camera's display (here, Reversi). A battery indicator keeps you informed about the remaining power for shooting.

3 Installing the CHDK

3.1 Requirements

Installing the CHDK requires you to prepare the camera's SD card and install the CHDK onto that card. To do so, you need:

- An SD card with no more than 4 GB. Larger cards can be used with the CHDK on some cameras, but some features such as AUTORUN will not work. Therefore, it is better to stick with cards that have no more than 4 GB of memory[1].
- A suitable card reader. Modern notebooks are usually equipped with such a card reader. Alternatively, an external device can be connected to the USB port. They don't cost an arm and a leg.
- Some knowledge of navigating file systems and downloading and installing software.

3.2 Does a CHDK exist for my camera?

Each camera model requires its own build of the CHDK. To be more precise, each firmware version of the camera's operating system requires its own CHDK build. So, once you have established that your camera model is supported by the CHDK[2], you have to find out the firmware version of your camera.

Normally, your camera does not show this information. It must be enabled to do so, and this is done by placing a small file named `ver.req` (for version request) into the root directory of the memory card. On Windows systems, this is easily done by inserting the card into a card reader, downloading the program *CardTricks* (http://chdk.wikia.com/wiki/CardTricks), and formatting the card as FAT (provided that you use a card with no more than 4 GB of memory). If you have important content on the card, make sure to copy it to other media first!

1 A discussion about using memory cards with more than 4 GB is found in the appendix.
2 The supported camera models are listed at http://chdk.wikia.com/wiki/CHDK.

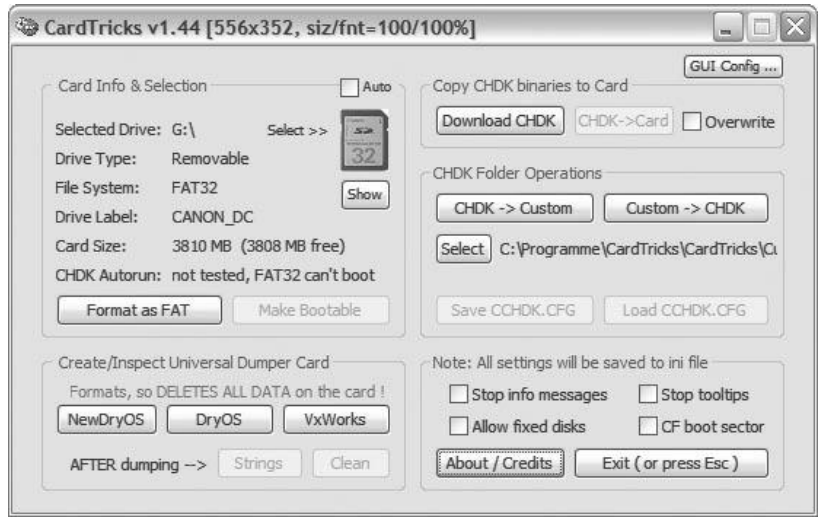


Figure 3-1
The *CardTricks* control screen. New cards usually come formatted in the FAT32 file system. They must be reformatted in the FAT file system. With large cards, the button **Format as FAT** will automatically change to **Format as FAT32**.

Figure 3-2
This is how the camera display should look after pressing FUNC/SET and DISP. By keeping FUNC/SET pressed and pressing DISP more than once, you can scroll down the screen to show additional information.

CardTricks will create the file `ver.req` during that process. Now you may insert the card into the camera. Then:

1. Switch the camera off.
2. Switch the camera to *Playback Mode*.
3. Switch the camera on again.
4. Press FUNC/SET. While keeping FUNC/SET pressed, press DISP. Now you should be able to read the firmware version number from the screen. It should be something like `Firmware Ver GM1.01B`. If you get the clock instead, you were not quick enough in pressing the DISP button.

If this does not work, try to rename the file to `vers.req`. Some newer cameras work with this file name.

3.3 Downloading the CHDK

Now go back to the *CardTricks* utility and press the **Download CHDK** button. This will launch your web browser and open the URL http://mighty-hoernsche.de/. Find your camera and the firmware version, and click on the link to start the download.

The website actually lists two downloadables for each camera and each firmware version—a complete bundle with CHDK, example scripts, fonts, and other goodies; and a smaller version with just the core CHDK. Don't hesitate to download the larger version—the difference of 200 KB is hardly noticeable even on the tiny 256 MB cards supplied with your camera.

Before the downloaded software can be transferred to your camera, you should make your memory card bootable. Just reinsert the card into the card reader and press the **Make Bootable** button in *CardTricks*.

Now you may transfer the CHDK to your card. Press the button **CHDK > Card** and select the downloaded file. *CardTricks* will simply unzip this file into the card.

Remove the card from the card reader. Set the write protection by moving the little slider away from the contacts. This will enable the AUTORUN feature. Don't be afraid, the camera will still be able to write pictures to the card. The CHDK will see to that.

Figure 3-3
4 GB SD card with the slider in write-protection position.

Finished. Keep your fingers crossed and insert the card into your camera. Switch the camera on. Shortly after the Canon splash screen, the CHDK splash screen will appear. If this is the case, you have successfully installed the CHDK on your Canon camera.

3.4 Manual installation

CardTricks is a *Windows*-based program. If you work with a Mac[3] or with Linux, you will not be able to use it (though it might be possible through an emulator). But there is another method to get the CHDK up and running:

1. First, format the card with the FAT16 file system. This must be done with a card reader and a computer—the camera would format your card with FAT32. (On the Macintosh, invoke the terminal program and enter "df" to start the harddisk manager. Then enter a formatting command such as "newfs_msdos -F16 /dev/*cardname*" where "*cardname*" is the volume name of your card.)
2. Then, make an educated guess at your firmware version. Download the corresponding CHDK binaries and unpack them to your card. Again, you need a card reader to do so.

3 A script for easy installation on the Mac is found on http://chdk.wikia.com/wiki/FAQ/Mac.

Figure 3-4
The `Firm Update ...` function of the camera can be used to load and activate the CHDK.

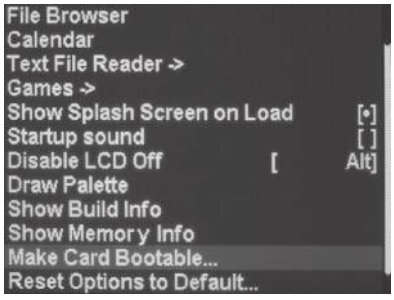


Figure 3-5
The CHDK submenu *Miscellaneous*. The entry *Make Card Bootable...* does exactly what it says.

3. The distribution contains a file `ver.req` or `vers.req`. This allows you to determine the real firmware version as described in section 3.2. If it does not match your guess, download the correct version and unpack it into your card.
4. Now you are ready to boot manually. Leave the camera in *Playback Mode* and insert the card. Press **MENU** and navigate down to the last item (*Firm update...*). Press **FUNC/SET**, select the *OK* button, and confirm again with **FUNC/SET**. The CHDK is loaded and will soon show the CHDK splash screen. It stays loaded until the camera is switched off. Actually, no firmware upgrade was performed! The *Firm update...* function was just hitchhiked by the CHDK.
5. To enable the AUTORUN function, you must make your card bootable. That can be done with the CHDK, too. Press the **ALT** key (chapter 4) followed by the **MENU** key to invoke the CHDK menu. Navigate to *Miscellaneous stuff* > *Make Card Bootable...* and press **FUNC/SET**.
6. Switch the camera off, remove the card, and lock the card to enable AUTORUN mode. Insert the card again and switch the camera on. CHDK will now boot automatically each time the camera is switched on—until you disengage the write protection lock.

Once you have installed the CHDK on your card, you should, as a matter of fact, avoid formatting your card to get rid of images. Formatting would also delete the CHDK and require a complete reinstallation. Instead, you have two options for deleting obsolete pictures:

- Use the camera's original delete functions to delete images. Switch the camera to *Playback Mode,* press **MENU,** and select *Erase...* Then you can delete by selection, by folder, or all images. There is a hitch: the camera will only show JPEG images or folders containing JPEG images. RAW and DNG images obtained through the CHDK (section 4.5) are not shown and not deleted.
- Use the file browser provided by the CHDK (section 4.10.1). Press **ALT** (chapter 4) followed by **MENU**. Then select *Miscellaneous Stuff* > *File Browser* and navigate to the folder that you want to delete. Press **DISP** and answer the prompt with **Yes**.

3.5 The warranty question

The big question for many camera owners is: what about the manufacturer's warranty when I install the CHDK in the camera? We don't think that there should be a problem. The CHDK is installed on the memory card.

The camera is obviously designed to load and execute software found on the card—the situation is similar to fitting a third-party lens to a DSLR.

The CHDK alters neither the camera nor the camera's firmware in any way. Reformat the card (or use a fresh card), and the CHDK is gone. The camera continues to run under its original firmware. In fact, before sending in the camera for warranty, you should remove any memory cards from the camera. A camera equipped with a CHDK-enabled card may disturb service technicians and make their job difficult. In addition, the card may also contain images that you don't want to share with others.

4 Teach Your Camera New Tricks

We have already mentioned the ALT key. Use it to switch the camera into *<ALT>* mode (or CHDK mode) where all CHDK menus and scripts are accessible. Note that while in *<ALT>* mode, none of the keys (including the shutter release) work in the traditional way. To return to normal operation, you must leave *<ALT>* mode. This is done with another click on the ALT key. *<ALT>* mode is indicated by a small <ALT> sign at the bottom of the display (blue in *Recording Mode* and white in *Playback Mode*).

Now, where is this mysterious key? There is no key on your camera labeled ALT. In fact, the actual key acting as the ALT key varies from camera to camera:

- *A* series and *SD* series: the Print button is used. A short press (like a click) is sufficient; a long press will either have no effect or invoke a native camera function.
- *G* series and *S* series: the Shortcut button is used by default. Actually, the *ALT* function can be assigned to the Shortcut, Flash, Timer, ISO, or Video buttons.

4.1 Using menus

Invoking the CHDK menu is almost as simple as invoking the native menu. Switch to the *<ALT>* mode by pressing the ALT button, and then press the MENU button. You can exit the menu by pressing the MENU button again or by pressing the ALT button, which will also leave *<ALT>* mode and return to normal mode.

Some menus are too large to be displayed on the screen. In this case, the menu shows a scroll bar at the right-hand side. Use the UP and DOWN buttons to scroll to the desired position.

Menu items are either parameters or submenus. A right arrow identifies submenus. To open a submenu, press the FUNC/SET button. To return from a submenu to the higher menu level, navigate to the last item on the screen which should be named **Back**. Press FUNC/SET on that item. Alternatively, you can press the DISP button to return to the parent menu.

Pressing the LEFT, RIGHT, or FUNC/SET buttons changes values in the other menu entries. The change immediately becomes active—an extra

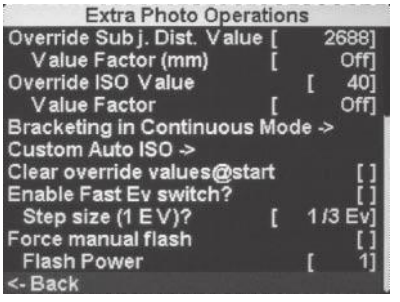


Figure 4-1
The *Extra Photo Operations* submenu. The scroll bar on the right-hand side indicates that the menu is scrolled down to the bottom. The last entry, the **Back** entry, leads back to the main menu. This submenu has two child submenus: *Bracketing in Continuous Mode* and *Custom Auto ISO*.

confirmation is not required. Simply leave the menu by pressing the MENU or ALT button when you are finished.

Some entries have a large value range and dialing in the desired value with the LEFT or RIGHT button would be very tedious indeed. Therefore, the CHDK programmers have equipped such entries with a child entry named **Value Factor** that usually has the values `Off`, `1`, `10`, `100`, etc. The value selected here is multiplied by the value of the parent entry. For example, if you dialed in a value of **40** for the entry *Override ISO Value* (Figure 4-1) and selected a value of **10** in the child entry *Value Factor*, you would be specifying an ISO speed of `40*10 = 400`. Selecting the value **Off** in the *Value Factor* disables the parent entry. In this example, selecting `Off` for *Value Factor* would switch off the manual override of the ISO speed and return to the camera's native ISO speed control system.

4.2 Customizing the user interface

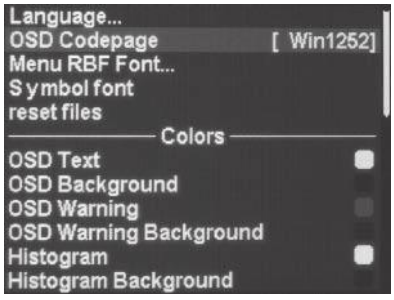


Figure 4-2
The *Visual Settings* submenu. The first entries are dedicated to language, code page, font, and symbols, while the rest of the entries are used to specify the text and background colors of the different information items. Note that the font settings apply to menus only, not to OSD text.

The CHDK has many functions to customize the user interface. Most of these functions are found in the submenu *Visual Settings*. Here you can change the language of the CHDK menus. (The language of the native camera menus is changed separately. Consult your camera's user manual to do so.) You can change the *OSD Codepage* to allow for different national characters. You can also select between plain, bold, serif, and sans serif fonts in different sizes, and switch between differently sized symbols (10 or 16 pixels).

4.2.1 OSD Codepage

By default, the CHDK is set up with the OSD Codepage `Win1251` that supports the Cyrillic alphabet (the CHDK was originally created by a Russian programmer).

For English, Spanish, or other Western languages, switch the codepage to `Win1252`. Invoke ALT > MENU > *Visual Setting* > *OSD Codepage...* and press FUNC/SET to select `Win1252` (Figure 4-2).

4.2.2 Fonts

By default, the CHDK uses the smallest font available for its menus, thus maximizing the number of menu items displayed on one page. Depending on your eyesight, you may want to select a larger font. For me, this has very much increased the joy of using the CHDK. Invoke ALT > MENU > *Visual*

Setting > *Menu RBF Font...,* press FUNC/SET and select a font that suits you, or press MENU to escape.

Note: Not every font supports all national characters. If the national characters of the chosen language do not appear correctly, select a different font. You may have to repeat this process a few times until you find a satisfying solution.

Hint: To switch back to the default font, just select a non-font file.

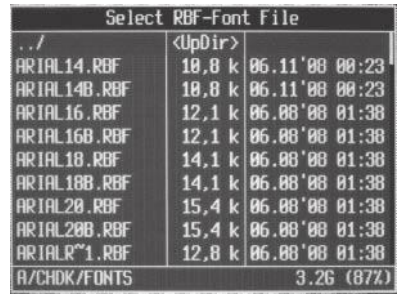


Figure 4-3
The CHDK file browser is used to select a font for displaying menu items.

4.2.3 Colors

The rest of the submenu *Visual Settings* is dedicated to colors. Both the text color and the background color can be changed for different information items. The default background color is a semi-transparent, neutral gray. To change a color, press FUNC/SET on the color entry. This brings you into the color chooser. Select a suitable color with the LEFT, RIGHT, UP, and DOWN buttons. Commit the selection with FUNC/SET or cancel with MENU. Both transparent and opaque colors are available.

4.2.4 Organizing the screen

The amount of information the CHDK is able to show on the display is amazing—almost any camera state can be displayed on the screen. Therefore, it becomes mandatory to select the information carefully and to arrange its layout properly. Otherwise, you would be overwhelmed by an information overload that would be disturbing in shooting situations instead of being helpful. Most of these settings are found in the submenu *OSD*.

Let's discuss these settings one by one:

- *Show OSD*. Here you can specify whether you want to show the CHDK OSD information at all. Note that this setting does not affect the native OSD information. It also does not affect the *Edge Overlay,* the *Histogram,* or the *Zebra* function. Simply press FUNC/SET to switch the CHDK OSD on or off.
- The corresponding entry *Hide in?* specifies when to show the OSD information if it is switched on. Select **Don't** if you do not want to hide the OSD at all, select **In Playb** if you want to hide the CHDK OSD during playback mode, select **On Disp** if you want to hide the CHDK OSD together with the native OSD (which probably makes the most sense), or select **both** if you want to hide it in both situations. Unfortunately, this setting does not affect all CHDK OSD information items. The only way

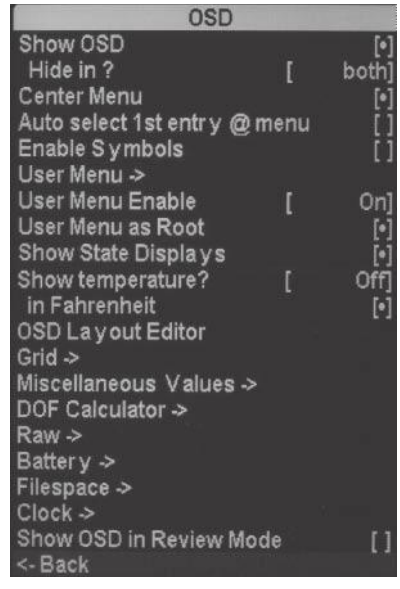


Figure 4-4
The OSD submenu. This image was compiled from two subsequent screenshots.

to get a completely clean screen is to switch off the CHDK OSD completely (see above).

- *Center Menu*. Displays the CHDK menus in the center of the screen if switched on. If switched off, the CHDK menus are aligned to the top of the screen.
- *Auto select 1st entry @ menu*. When switched on, the first item of a freshly opened menu is selected automatically. If switched off, no item will be selected.
- *Enable Symbols*. Determines whether symbols should be shown at the beginning of each menu item. I find the information value of these symbols rather limited and I can read the menu easier without the symbols.
- For the *User Menu* entries, please see section 4.2.5.
- *Show State Displays*. Determines whether the settings made in the submenus *Extra Photo Operations* (section 4.3.1) and *Custom Curves* (section 4.3.8) should be displayed on screen.
- *Show temperature*. Determines if and when temperature should be displayed on screen (**Off**, **Optical**, **CCD**, **Battery**, **All**). This can be useful when operating in a very hot or cold climate. (The optical temperature is the temperature of the lens elements and is the closest one to the environment temperature.) The temperature reading is in degrees Celsius if the following menu entry *in Fahrenheit* is not switched on.
- *OSD Layout Editor*. This entry allows you to modify the layout of the OSD information items on the screen. For this purpose, the CHDK organizes the information items into groups: histogram, DOF calculator, exposure status display, remaining RAW images, miscellaneous values (focal length and other values), battery symbol, memory card symbol, vertical memory usage bar, horizontal memory usage bar, battery text, memory usage text, clock, temperature, remaining video time, EV correction factor photo, and EV display video. By pressing the FUNC/SET key repeatedly, you can select between these groups in the given sequence. The selected group is shown with a green frame. To modify the position of the selected group on the screen, simply press the LEFT, RIGHT, UP, and DOWN buttons. You can leave the editor with the MENU button.

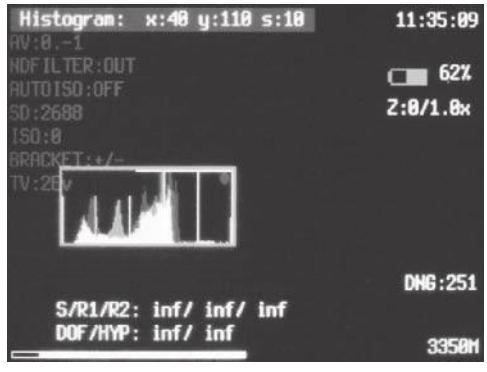


Figure 4-5
The OSD layout editor in action. The status bar (top left) displays the currently selected layout item (Histogram), its x/y-position, and the step width(s). The step width can be changed between 1 and 10 via the DISP button.

- *Grid*. See section 4.2.6 for this submenu.
- *Miscellaneous values*. See section 4.2.7 for this submenu.
- *DOF Calculator*. See section 4.2.8 for this submenu.
- *RAW*. The *RAW Showing* submenu controls the display of information related to shooting in RAW format. The entry *Show RAW state* controls whether RAW-related information is shown at all. If yes, the subentry *Show RAW shoot remain* controls whether the number of remaining RAW images is shown. A warning is given if this number falls below the value specified in the subentry *Warning threshold*.
- *Battery*. The *Battery Showing* submenu controls the display of battery-related information such as symbol, level (percent), and current voltage. In addition, it allows you to set reference values for the maximum and minimum voltage. When set up correctly, the level will move within the full range from 0 to 100 percent. To use this function, first enable *Show Battery Voltage*. Now you can determine the voltage of a freshly loaded battery and the voltage of the battery shortly before the camera powers off due to battery exhaustion. When you have found out these values, you may want to dial them in under *Battery MAX Voltage* and *Battery MIN Voltage*.
- *Filespace*. The *Filespace showing* submenu controls the display of memory space information. It determines whether to show a filespace icon and a vertical or horizontal space bar. The length and thickness of the bar can be controlled in the subentries *Size on screen* and *Width/Height*. The free space in the memory card can be shown as a percentage of its total capacity or as an absolute value in MB. Again, warnings can be given if this free space falls below a defined threshold, which can be specified in both percentage and MB.
- *Clock*. The *Clock* submenu controls the display of the system clock. The entry *Show Clock* can be switched to **Don't**, **Normal** (hh:mm), and **Seconds** (hh:mm:ss). The *Clock format* can be switched to 24h or 12h format, and the *12h clock indicator* can be switched between **PM**, **P**, and a period. The entry *@Shutter halfpress show* controls if and how the clock is displayed when the shutter button is half-pressed (**Don't**, **Full**, **Seconds** (only)).
- *Show OSD in Review Mode*. This submenu controls whether OSD information is shown in *Review Mode*. (*Review Mode* is activated when you hold the shutter button down after a shot and press FUNC/SET.) On my SD1100, however, this feature seemed to be broken; the OSD information was always shown in this mode, independent of the settings.

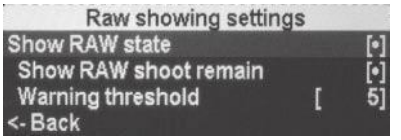


Figure 4-6
The *RAW showing* submenu

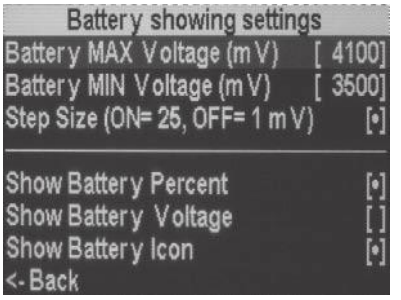


Figure 4-7
The *Battery showing* submenu

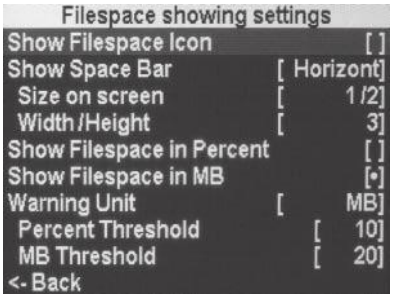


Figure 4-8
The *Filespace showing* submenu

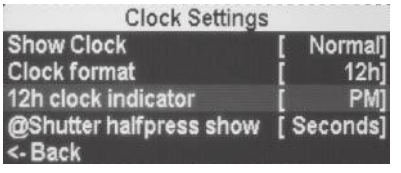


Figure 4-9
The *Clock* submenu

4.2.5 User menus

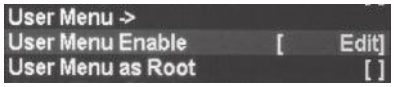


Figure 4-10
Two items control the configuration of the User Menu. Here, the user menu is switched to editing mode.

The number of entries in the CHDK menu is quite overwhelming. Searching the same items again and again and stepping through the submenus can be quite tedious. The CHDK therefore offers the option to organize the most important menu items (favorites) into a user-defined menu. This menu is, by default, accessible through ALT > MENU > *OSD Parameters* > *User Menu*. If you try this, you will find that the user menu only contains a single item, *Main Menu,* that leads back to the main menu. This item is always the first item in the user menu.

Now, how do you configure your own menu? It's quite simple: first, you switch the user menu into editing mode. Then you add menu items from the main menu by selecting them:

1. Set the entry *User Menu Enable* to **Edit**.
2. Navigate to the menu item that you want to add to the user menu.
3. Half-press the shutter button to add this item to the user menu. On some cameras the FUNC or ERASE button is used instead.
4. Repeat steps 2 and 3 for more entries.
5. To remove menu items from the user menu, navigate to the user menu, then half-press the shutter button (resp. FUNC or ERASE) on the item to be deleted.
6. In the menu *OSD Parameters,* set the entry *User Menu Enable* to **On** or **OnDirect**. **OnDirect** will automatically launch the user menu when you switch to the *<ALT>* mode, saving you a click on the MENU button. The MENU button, however, can still be used to launch the main menu or to leave all menus. Alternatively, you can set *User Menu Enable* to **On** and enable the option *User Menu as Root*. This will immediately launch the user menu when you press the MENU button in *<ALT>* mode. Another click on MENU will close the user menu. With this option set, the main menu can only be reached through the first entry in the user menu.

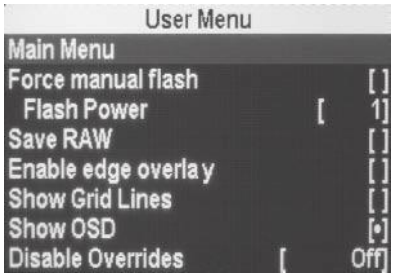


Figure 4-11
A configured user menu. The first item always leads to the main menu; all other items are favorites. As a result, you can change the most important settings very quickly.

4.2.6 Grids

In the analog days, an exchangeable ground glass was mostly a feature of professional cameras. Today, many professional and semi-professional DSLRs still offer the option of exchanging the ground glass. Some of these screens come with etched grids, which can be especially useful for architectural work and micro- or macrophotography.

With digital compact cameras and their electronic viewfinders, grids can be easily shown on the camera's display. Most of these cameras come with a few built-in grids. For example, my SD1100 offers the option of displaying 3:2 guides and a 1/3-rule grid. The 3:2 guides allow for easier

judging of the outcome if you plan to print in 6" x 4" format, for instance. The 1/3-rule grids are mostly used for positioning the horizon in landscape photography; dividing the image by a ratio of 2:1 or 1:2 is more pleasing than placing the horizon in the center.

The CHDK greatly extends the range of available grids. You can even define your own grids in the form of a simple text file. Each grid is implemented as a `.grd` file in the folder `CHDK/GRIDS/`. If you want to roll your own, you will find the necessary instructions in the file `README.TXT` in the same folder. Predefined grids are:

- **3to2grid**. Indicates the 3:2 crop ratio, e.g., for 6" x 4" prints.
- **3to8grid**. Organizes the image horizontally and vertically into 3:2:3 divisions. This is very close to the *Golden Ratio*. It can be used to compose an image in a visually pleasing way by aligning the prominent structures with the grid lines.
- **golden1**. Indicates the "Golden Point". The most important subject in the image should be placed where the two lines meet. Horizontal or vertical reflections of this grid can be used, too, resulting in four "golden" points.
- **golden2**. Identical to **golden1** but showing two "golden" points.
- **id**. Based on the ISO/IEC 19794-5:2005 standard for administrative identity photos (passports, identity cards, driving licenses, etc.). The large red ellipse shows the maximum head size, the small green ellipse the minimum head size. The gray area is where the eyes have to be.
- **rulecross** (**rulecr~1** on DryOS). A grid typically used in aerial photography and micro/macro photography.
- **rulers**. Similar to rulecross, but without the crosshair pattern.
- **sports**. A grid simulating a sports finder. The nested frames represent the different cropping lines. The idea is that you shoot at the highest resolution and later crop along the lines shown in the grid—resulting in a telephoto shot with lower resolution. The main advantage over digital zoom is that the display shows the events outside the cropping area, too. This gives you time to prepare for the shot and enables you to catch the event in time.
- **stolen**. A grid for the paranoid. When activated, the grid darkens the screen showing only the words "STOLEN CAMERA!" The idea is that a person unaware of the CHDK will not be able to use the camera. A new memory card, however, would cure it.
- **third_h**. Organizes the image horizontally into equally sized divisions (horizontal 1/3-rule).
- **third_v**. Organizes the image vertically into equally sized divisions (vertical 1/3-rule).

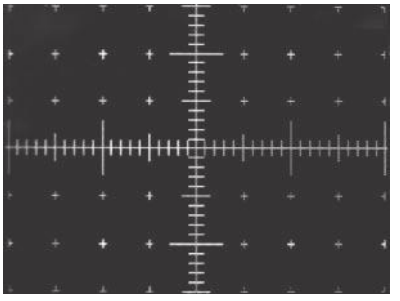


Figure 4-12
The **rulecross** grid

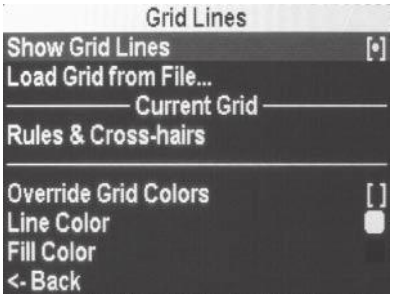


Figure 4-13
The *Grid Lines* submenu. Normally, grid files define the colors for lines and background. However, by enabling *Override Grid Colors* you can override these colors with your own color selections.

To load one of these grids, invoke the entry *Load Grid from File...* . This will launch the file browser (section 4.10.1). Select the respective grid file using the UP and DOWN buttons, and then press FUNC/SET to commit or MENU to abort. The grid may be switched on or off with *Show Grid Lines*.

4.2.7 Miscellaneous Values

The *Miscellaneous Values* submenu combines settings for the display of values related to focus, zoom, and exposure. The first entry, *Show Misc Values,* determines when these values are shown: not at all (**Don't**), always (**Always**), or when the shutter button is half-pressed (**Shoot**). The entry *Show values in video* can be set to show these values while the camera is in video mode.

Now, let's have a look at the various information items:

Figure 4-14
The *Miscellaneous Values* submenu. This image was compiled from two display screens.

- The *Zoom Value* can be shown as the zoom factor (**x**), as the true focal length (**FL**), or as the 35mm equivalent focal length (**EFL**).
- If you are using a lens converter, such as a telephoto converter or a wide-angle converter, you can dial its factor into the entry *Adapter Lens Scale*. For example, if you use a telephoto converter with a factor of 1.75, dial in the value 175. This setting will be used to display the focal length or the zoom factor correctly, and also to compute the depth of field correctly (section 4.2.8).

The remainder of the display options are exposure-related. The exposure system of Canon cameras is engineered along the APEX system *(Additive System for Photographic Exposure)*. This system defines several exposure-related entities and puts them in relation to each other:

- *Time value (Tv)* represents the shutter speed.
- *Aperture value (Av)* represents the lens aperture.
- *Exposure value (Ev)* represents the joint effect of exposure time and aperture.
- *Speed value (Sv)* represents the sensitivity of the sensor (the ISO setting).
- *Brightness value (Bv)* represents the scene brightness.

Internally, all these values use a logarithmic scale. This makes it easy for the camera to derive values from each other. The relationship is simple:

```
Av+Tv = Bv+Sv = Ev
```

When in auto mode, the camera determines the entities *Av, Tv,* and *Sv* from the measured scene brightness *(Bv)*. Depending on the program chosen

(Landscape, Sports, etc.), the camera will determine the optimal compromise between those entities. In manual mode, most cameras allow you to set a fixed ISO value *(Sv),* and some cameras even allow you to select a fixed shutter speed *(Tv)* or a fixed aperture *(Av)*. In such cases, the camera can easily derive the other values from the above formula.

Let's now take at look at the remaining parameters in the *Miscellaneous* group and their abbreviations when displayed:

Value Hg	In Misc
"Real" aperture The true aperture, not rounded for display purposes.	Av
"Real" ISO The true sensor speed as used in the APEX formula.	I-R
"Market" ISO The sensor speed as displayed by the native Canon firmware. This value is rounded and approximately 1.6 times higher than the real ISO value, probably for marketing purposes. Therefore, the nickname *Market ISO*.	I-M
Set Exposure Ev The effective *Exposure Value* computed from shutter speed and aperture (Tv+Av).	Evs
Measured Ev The *Exposure Value* computed from scene brightness and sensor speed (Bv+Sv).	Evm
Set Bv The *Brightness Value* computed from shutter speed, aperture, and sensor speed (Tv+Av-Sv).	Bvs
Measured Bv The measured scene brightness.	Bvm
Overexp. Value Shows an overexposure value computed from the APEX values: Av+Tv–(Bv+Sv). This is without flash illumination.	dE
Canon Overexp. Value Shows an overexposure value computed by the native Canon firmware. This is a rounded value.	dEc
Scene Luminance Shows the scene brightness in *Candelas* per square meter.	B

The differences between set *Ev* and measured *Ev,* and between set *Bv* and measured *Bv,* are usually small when the camera is in automatic mode. However, if you use *Overrides* (section 4.3.1), the differences can be

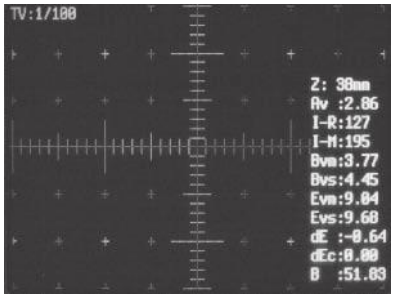


Figure 4-15
The *Miscellaneous Values display* (right) with all entries enabled. At the top is the 35mm-equivalent focal length (Z). Here the screen shows a 35mm-equivalent of 38 mm – the true focal length of the lens, however, is 6.2 mm. The dE entry indicates that the exposure is slightly incorrect. The reason is that a manual shutter speed of 1/100 sec was set via an *Override* (top left).

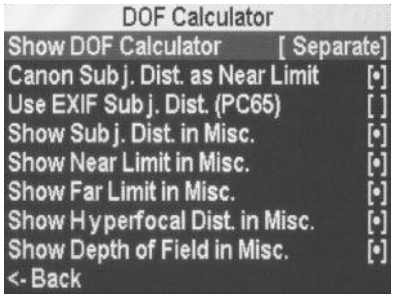


Figure 4-16
The submenu *DOF Calculator* is used to configure the DOF Calculator.

substantial and the *Overexposure Value* will be significant. When using *Overrides,* you should definitely display at least the *Overexposure Value*.

The small differences in automatic mode are caused by the fact that Canon software quantizes all values to 96 units per f-stop. This can lead to small (and insignificant) rounding errors.

4.2.8 Customizing the DOF calculator

The *Depth of Field* (DOF) calculator is a useful tool built into the CHDK. Based on the real aperture value and the focal length, it is able to compute values such as near sharpness limit, far sharpness limit, hyperfocal distance, and depth of field.

The entry *Show DOF Calculator* allows switching off the DOF calculator altogether (**Don't**), displaying its values in a separate group (**Separate**, see Figure 4-16, or displaying the values within the *Miscellaneous Values* group (**In Misc**). Depending on these settings, the displayed values are represented differently:

Value	Separate	In Misc
Subj.Dist. Subject distance as determined by the AF system or as set manually.	S	SD
Near Limit The near limit of sharpness.	R1	NL
Far Limit The far limit of sharpness.	R2	FL
Hyperfocal Dist. The minimum distance that must be set to achieve sharpness until infinity. The near limit is the hyperfocal distance divided by 2.	Hyp	Hyp
Depth of Field The difference between the near and far limits.	DOF	DOF

A bit disturbing is the fact that you can choose between different distance values:

- The option *Canon Subj. Dist. as Near Limit* uses the distance determined natively by the camera as the near limit. Based on this value, the subject distance and the far limit are both computed by the CHDK. My observation is that this option results in values that are too large.

- If this option is not set, the distance determined natively by the camera is used as the subject distance, and the near and far limits are computed. The result seems only slightly too large.
- The option *Use EXIF Subj. Dist. (PC65)* uses instead the distance written into the EXIF data for those computations. This is the same value as in *Property Case 65* (245 for DryOS, section 5.5.12). According to my observation, this value is too small.

Altogether, none of these three settings is precise. I get the best result with both options switched off, but you must find for yourself the best setting for your own camera.

4.2.9 Other user interface options

More useful options for customizing the user interface are found in **ALT** > **MENU** > *Miscellaneous Stuff:*

- *Show splash screen on load*. Can be used to hide the splash screen. At times, this screen can be a nuisance, especially if you are in a hurry to take a photo immediately after switching on the camera. If hidden, the only visual indication of the CHDK booting is a short flash of the blue LED.
- *Startup sound*. A short beep can be used to indicate that the CHDK has booted up.
- *Use zoom buttons for MF*. This entry is only available on cameras with manual focus. It allows a more precise setting of the manual focus by using the zoom buttons.
  - *A-Series:* With manual focus enabled, the zoom rocker mimics the left/right controls. You can adjust anything that is controlled by the left and right directional controls via the zoom rocker—such as manual focus, shutter speed, or aperture. Zooming is not possible in this mode.
  - *S-Series:* While the **MF** button is held down, the zoom rocker will control manual focus.
- *Disable LCD Off*. Select **No** if you always want the display to turn off after the time specified in the *Power Saving* section of the native camera settings. Select **Script** if you don't want the display turned off while a script is running. Select **Alt** if you don't want the display turned off while the camera is in *<Alt>* mode. This, of course, includes **Script** mode.

4.3 Exposure

Most compact cameras come with automatic exposure only. Usually they support a variety of automatic program modes such as portrait, landscape, sports, kids&pets, night shot, sunset, snow, etc. However, the smaller and lower-priced models usually do not feature aperture priority, exposure time priority, or a fully manual mode. There are simply no provisions to dial in an aperture value or a shutter speed. If they offer a manual mode, it only allows dialing in an exposure correction value, and that's it.

The CHDK has set out to remedy this situation. It allows overriding the values determined by the camera's exposure system with values dialed in by the user. This is possible for aperture (or neutral density filter), exposure time, sensor speed, and flash power. In addition, the CHDK offers a highly customizable *Auto ISO* system and advanced features for exposure control, such as a live histogram and zebra areas. We will discuss those features in the following sections.

4.3.1 Overrides

As already mentioned, the CHDK allows overriding aperture, exposure time, ISO speed, and flash power. This works in all camera modes, automatic and manual. So, for example, if the camera is in automatic mode and you override the exposure time only, the camera will automatically choose the right aperture; the camera will be in *Shutter priority (Tv)* exposure mode.

But not all cameras can do this. Cameras equipped with an ND filter instead of a diaphragm cannot adjust the aperture and completely revert to manual mode. In this case, you must adjust the ND filter state and the ISO speed yourself to obtain a properly exposed image. The density of the ND filter depends on the maximum aperture—the ND filter simulates a setting of f/8 at the focal length at maximum aperture. So, if the maximum aperture is f/2.8, the density of the ND filter will be three f-stops. With a telephoto setting where the aperture is perhaps only f/4.9, the aperture simulated by the ND filter would be around f/14. With the CHDK in place, you can normally keep the ND filter out and use very short shutter speeds when conditions become too bright. An ND filter makes sense if you need rather long shutter speeds for compositional reasons (waterfalls, architectural images, etc.).

So, with diaphragm-less cameras your options for exposure control are somewhat limited. You can't just open or close the aperture to compensate for variations in shutter-speed. Instead, you must adapt the sensor speed. For example, if the camera computes a shutter speed of 1/100 sec and you override it with 1/1000 sec, you have to make up for the 3 1/3 stops less

light caused by the shorter exposure time by increasing the sensor speed—let's say, from 50 to 500. Manually, this is not a problem. You just dial the new ISO speed into the *Overrides*. Automatic operation with shutter priority, however, is only possible on cameras with ND filters when using a suitable script for exposure control (section 5.7.4).

Let's return to cameras that do have diaphragms. On such a camera, if you override the aperture only, the exposure system will select a suitable exposure time; the camera will be in *Aperture-priority (Av)* exposure mode[1].

All of the *Overrides* are combined in the submenu *Extra Photo Operations*. The first entry of that submenu is used to switch the camera between native mode and override mode.

- **Disabled**. The *Override System* is completely disabled.
- **Off**. All *Override Values* are disabled, but the display shows the remark NO OVERRIDES.
- **On**. All *Override Values* are active. The display, too, shows the override values if option *OSD > Show state displays* is enabled (section 4.2.4).

Because stepping through the menus can be a bit awkward, the CHDK offers a key combination to toggle the *Overrides* between **On** and **Off**. This key combination depends on the camera type:

- Cameras without **DEL** key and SX100: **Shutter halfpress + LEFT**
- G7: **Shutter halfpress + UP**
- All other cameras: **Shutter halfpress + DOWN**

Optionally, the *Overrides* state *(On* or *Off)* can also be applied to the *AutoISO* feature (section 4.3.2) and the *Bracketing* feature (section 4.6). This is controlled through the entry *Include AutoIso & Bracketing*. If the option is set, *AutoISO* and *Bracketing* will be enabled if *Disable Overrides* is off, and disabled if *Disable Overrides* is on.

Individual entries can be switched on or off through their *Value Factor* subentry. If not set to **Off**, this value factor is multiplied by the value of the main entry. For the *Shutter Speed* entry, there is another, more convenient possibility: when the entry *Shutterspeed enum type* is set to **Ev Step**, the only possible value factor is 1 (and **Off**). The *Shutter Speed Value* will step through all available shutter speeds by 1/3 f/stops (`2048 sec`, `1625 sec`, ..., `1.3 sec`, `1.0 sec`, `0.8 sec`, ..., `1/80000 sec`, `1/100k sec`) as we are used to in mainstream photography.

Otherwise, if the entry *Shutterspeed enum type* is set to **Factor**, the factor is simply multiplied by the specified shutter speed value. This method is

1 Again, this is not possible on cameras without diaphragms—you simply won't be able to override an aperture value.

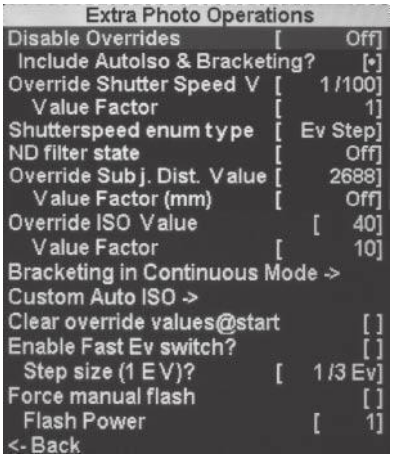


Figure 4-17
Overrides for exposure and other settings are controlled through the *Extra Photo Operations* submenu. An Aperture value override is not shown here because this camera has only an ND filter and no diaphragm.

better suited for scientific work. Of course, the real limits[2] of this range are defined by the camera hardware, but see sections 4.3.6 and 4.3.5 for very long and very short exposure times!

The same applies for ISO and aperture settings. For example, dialing in an ISO value of 6400 will have no effect if your camera only supports ISO 1600: the sensor speed, will be set to 1600. Remember, too, that in the case of the ISO speed, the *Override* value is the "real" ISO value—not the "market" value, which is approximately 1.6 times higher. So if you dial in an ISO value of 1000, the *Review Mode* will show an ISO of 1600.

For most cameras, the CHDK allows higher ISO values than what the native camera menu has to offer. On my SD1100, for example, the ISO scale ends at ISO 1600. The CHDK, in contrast, allows me to shoot images with ISO 6400 (dialed in as ISO 4000). Of course, you may rightfully ask, what about the noise? Aren't images shot at such high sensor speeds extremely noisy? In most cases, this is true, and it's probably the reason why manufacturers put an artificial upper limit on the ISO value. But there are situations where such ISO values can be useful. One example is when you use the averaging technique for hand-held night shots (section 4.5.6). In such a situation, a sharp and noisy image is more useful than an unsharp image with less noise. The noise can be remedied by averaging a dozen shots or so.

Even in the case of single shots, there are options. PC-based noise reduction programs such as *NoiseNinja* or *NeatImage* can do a better job than the camera does, and there are also some RAW converters with respectable noise suppression capabilities.

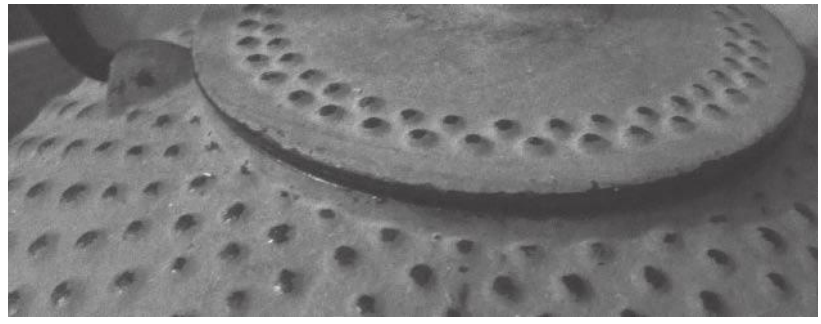


Figure 4-18
Japanese teapot detail. Photographed with *a Canon Digital Elph SD1100* as a DNG file. An ISO of 4000 was dialed in, the camera's reading was 6400, and the EXIF data recorded 6121. Exposure time was 1/20 sec (per Override), the aperture f/2.8 at a focal length of 6.2 mm (~ 38 mm). The red color of the teapot and the nearly monochromous character of the image kept the noise low because noise is dominant in the blue channel. Developed with *CaptureOne Pro* 5.0, which features excellent noise reduction.

Two more entries need an explanation:

- *Clear override values @start*. If this option is enabled, all values are reset to their default when the camera is powered up. Otherwise, the *Overrides* remain in place even after the camera is switched off.
- The option *Enable Fast Ev switch* turns the **UP** and **DOWN** buttons into exposure correction buttons. This makes sense because you don't have to step through the Canon menus to set an exposure correction value.

2 For the real limits of your camera, please see http://chdk.wikia.com/wiki/CameraFeatures.

Also, the available correction range is much larger than the two f-stops provided by the camera's native exposure correction. The step size can be controlled in increments of 1/6 f-stop through the subentry *Step size*.

On cameras where the **UP** and **DOWN** buttons are used for changing the ISO setting and for the self timer, however, there is a problem. With *Enable Fast Ev* set, these buttons will have two functions. Pressing the **UP** button repeatedly will change the *Ev* and the ISO speed; the **DOWN** button will change the *Ev* and the shooting mode.

The workaround is simply to wait a second after each click on **UP** or **DOWN**. After that time, the native function will be aborted so that only the *Ev* value is changed. To change the ISO value or the self-timer/series mode, simply click faster and afterwards correct the *Ev* value with the opposite button in delayed fashion. Or, temporarily switch off *Enable Fast Ev*. This less-than-optimal situation will likely change in future CHDK versions.

4.3.2 Custom Auto ISO

Apart from setting a fixed ISO value in manual mode, the native Canon exposure system provides two automatic ISO modes. These modes try to find a suitable compromise between sensor speed and exposure time when shooting in low light. The mode *ISO AUTO* puts the emphasis on image quality. It uses lower ISO values and longer exposure times and is therefore suited to subject matter without much movement, such as landscapes or portraits. In contrast, *ISO HI* puts the emphasis on speed. It uses higher ISO values (resulting in more noise) but shorter exposure times. It is therefore better suited to subject matter with fast movements, such as sports, children, animals, etc.

Of course, both of these automatic ISO modes are predefined by the manufacturer. The photographer has no chance to customize how they work. Since the camera does not know how steady your hand is or the minimal shutter speeds required, it can only make worst-case assumptions that may result in less-than-optimal ISO settings.

Enter the CHDK. The submenu *Extra Photo Operations* > *Custom Auto ISO* provides you with all the necessary controls to optimize the behavior of *ISO AUTO* and *ISO HI*.

Let's discuss the submenu entries one by one:

- *Enable Custom Auto ISO* is used to enable the parameters listed below. When disabled, the camera goes back to the native *ISO AUTO* and *ISO HI* mechanism.
- *Minimal Shutter Speed.* Here you can set a minimal shutter speed. The **Auto** setting can be used when your subject holds still, as in landscapes, portraits, or stills. For moving subjects, this setting will choose a

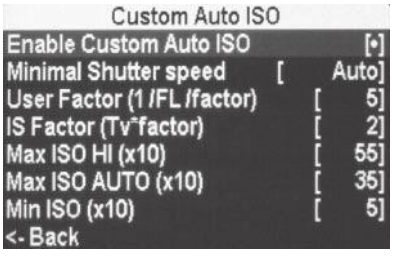


Figure 4-19
The *Custom Auto ISO* submenu

minimum shutter speed depending on the focal length and the factors specified in the following entries. The numeric settings (1/8, ..., 1/1000) specify a fixed minimum shutter speed. The camera will try to use this or a shorter speed. Typical values would be:
1/8–1/30 for slow movement
1/60–1/125 for general movement (kids, pets, etc.)
1/125–1/1000 for fast movement (sports, etc.)

- *User Factor (1/FL Factor)*. An old rule says that you should use the value of `1/focal_length` as your shutter speed to avoid camera shake. However, this rule was made for 35mm cameras. To adapt this rule to the optics of a digital camera, you need to specify the lens factor. To find out that factor, first switch the display of the focal length to **EFL** (35mm equivalent). This is done in *OSD* > *Miscellaneous Values* > *Zoom Value* (section 4.2.7). Now read the focal length (`Z`) from the display. Then switch the same entry back to **FL** (true focal length). Again read the focal length (`Z`) from the display. Divide the first value by the second. This is the start value for the *User Factor*. For example: on my SD1100 with its 1/2.5 inch sensor, I get an EFL of 38 mm and an FL of 6.2 mm in wide-angle setting. The resulting focal length factor is close to 6.1. Cameras with a 1/1.7 inch sensor have a focal length factor of 4.6.

  Now you can modify this value according to your ability to hold the camera steady. If you think that you can hold the camera quite steady, or if you are able to lean your body against some solid object, subtract 20 percent. If you are quite shaky, add 20 percent. When the camera is mounted on a tripod, dial-in the smallest value possible (1).

  When shooting in a vibrating environment such as an airplane, the *User Factor* (which is limited to an upper bound of 8) may not be suitable. Depending on the strength of vibrations, the above shutter speed formula must be modified to approximately `8/focal_length`. Then the *User Factor* must also be multiplied by `8`—too large to dial in. So instead, compute the minimal shutter speed from the EFL-reading (35mm equivalent) with the help of this formula and set it as a fixed value in *Minimal Shutter Speed*.
- *IS Factor (Tv* Factor)*. This factor takes the *Image Stabilizer* into account. An average *Image Stabilizer* will give you an exposure time extension of an approximate factor of 4 (2 f-stops) without risking shake, a poor *Image Stabilizer* about a factor of 2 (1 f-stop), and an excellent *Image Stabilizer* a factor of 8 (3 f-stops). Newer camera models usually have more powerful image stabilizers than older models. You can look into test reports (for example, on www.dpreview.com) to find out about the effectiveness of your camera's IS system. Or do some of your own measurements. **Hint:** Switch the *Image Stabilizer* off in environments with strong vibrations. It's built to correct hand shake, not vibrations with higher frequencies.

- The entries *Max ISO HI* and *Max ISO AUTO* are used to limit the maximum ISO chosen in the respective ISO mode, and thus to limit visible noise in the image. This value must not exceed the maximum ISO speed physically possible for your camera[3]. The values dialed in here are multiplied by 10, so, for ISO 800, dial 80.
- In *Min ISO* you should set the minimum ISO physically possible for your camera. Again, the values dialed in here are multiplied by 10, so, for ISO 100, dial 10.

4.3.3 Histogram

Your camera probably does have a histogram, at least in *Review Mode*. However, this histogram is only shown *after* an image has been taken.

The CHDK has a bit more to offer. One option is showing a *Live Histogram* on the display. The histogram can be shown when the shutter button is half-pressed (**Shoot**) or always. So, with a quick look at the histogram, you can decide whether the image is correctly exposed or whether it is necessary to adjust the exposure (see section 4.3.1 for the fast *Ev* buttons). In the case of too much contrast, however, it would be better to shoot an HDR bracketing series (section 4.6.2).

There is a lot to configure:

- Different histogram styles can be chosen in the entry *Histogram Layout* (Figure 4-21):

RGB	Displays the sum of the red, green, and blue pixel values.
Y	Displays the luminance component.
RGB Y	Displays RGB above luminance.
R G B	Three histograms, one for each color.
RGB all	Displays all five histogram types with RGB on top.
Y all	Displays all five histogram types with luminance on top.
Blend	Blends the three histograms of R G B into one.
Blend Y	Same as Blend, but with the luminance histogram added below.

- *Histogram Mode* allows you to choose between **Linear** and logarithmic (**Log**) scaling of the histogram Y-axis. Logarithmic scaling can be useful if the image contains a large amount of similar colors. With linear scaling, this would result in very high peaks.
- *When enabled, Show Histogram Over/Under EXP* displays **red dots** at both ends of the histogram if over- or underexposed areas are detected, as shown in some histograms in Figure 4-21.

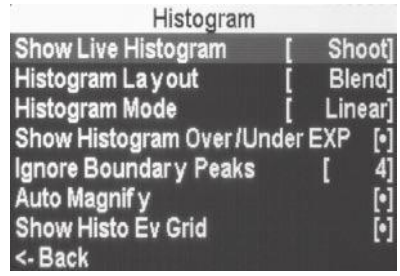


Figure 4-20
The CHDK histogram can be configured through submenu *Histogram*.

3 For the real limits of your camera please see http://chdk.wikia.com/wiki/CameraFeatures.

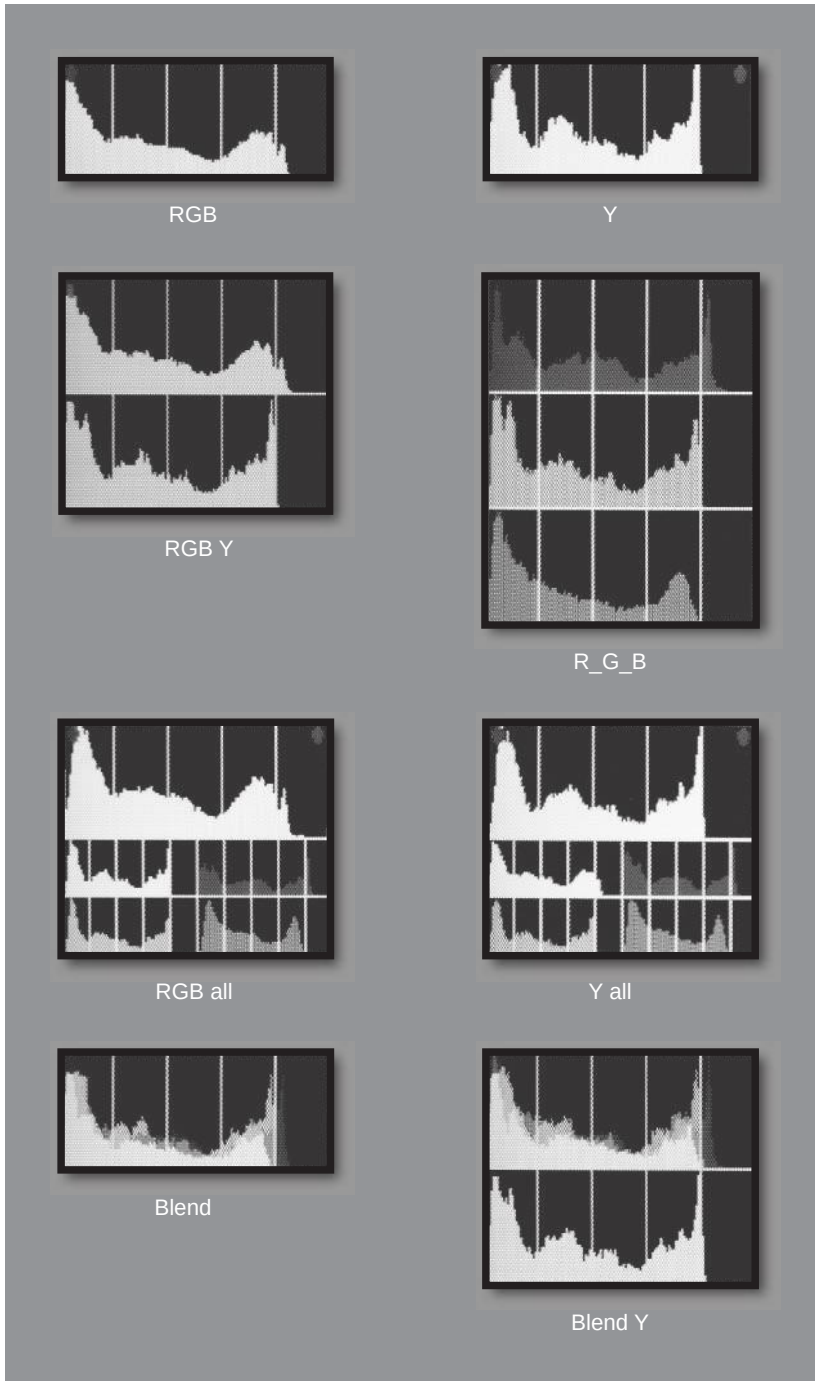


Figure 4-21
The different histogram styles

- *Ignore Boundary Peaks* allows clipping a specified number of pixels (0–32) at the left and right side of the histogram. Typically, this situation arises when the subject matter is small and placed on a monotonous background, as in night shots. A large black area would result in a high peak at the left side of the histogram, and the remaining information would vanish because of scaling. Removing these peaks makes the histogram readable again.
- Another way to deal with peaks is provided by *Auto Magnify*. If enabled and the histogram is filled to less than 20 percent, the Y-axis is enlarged and peaks are clipped. The clipped peaks are marked with a red dot, and the amount of magnification is displayed above the histogram.
- *Show Histo Ev Grid* adds four or five vertical grid lines to the histogram that indicate *Ev* values. The lines have a distance of one *Ev* (one f-stop).

4.3.4 *Zebra*

Another convenient way to control exposure is the *Zebra* feature. While the histogram informs you about the tonal range of the complete image, the Zebra feature identifies image areas that are under- or overexposed. These areas are displayed with a pattern overlaid on the image to make them stand out visually.

When enabled in the CHDK menu, Zebra areas are displayed by half-pressing the shutter. The feature also works in *Replay Mode* when half-pressing the shutter, allowing you to assess the image quality after the shot has been taken.

Again, there are various options to configure:

- *Draw Zebra* enables or disables the feature.
- *Zebra Mode* determines how the overexposed or underexposed areas are overlaid:

Blink 1	Solid overlay, blinks twice every second.
Blink 2	Solid overlay, blinks every second. This is my favorite. When there is something blinking on the screen, I take a closer look to see what's wrong, but the blinking is not so fast as to make me nervous.
Blink 3	Solid overlay, blinks every two seconds.
Solid	Solid overlay, no blinking.
Zebra 1	Striped overlay, thin diagonal lines.
Zebra 2	Striped overlay, fat diagonal lines.

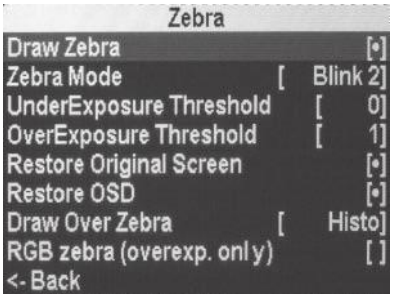


Figure 4-22
The *Zebra* submenu configures the Zebra mode.

- The sensitivity of the Zebra feature can be controlled by *UnderExposure Threshold* and *OverExposure Threshold*. The range goes from 0 (low sensitivity) to 32 (high sensitivity).
- The next two options only apply to the Zebra blink modes. When a Zebra pattern is drawn, both the Canon indicators and the CHDK indicators (for example, the DOF calculator) are erased by the pattern. To restore these information items during the blink pauses, set the options *Restore Original Screen* (for the Canon items) and *Restore OSD* (for the CHDK items).
- The option *Draw Over Zebra* applies to all Zebra modes. When this option is set to value **OSD**, all OSD information items will be drawn on top of the Zebra pattern and thus remain visible. When set to **Histo**, only the histogram is drawn on top of the Zebra pattern. When set to **Nothing**, the Zebra pattern will be drawn on top of all items. With a blinking Zebra pattern, I would rather set this parameter to **Nothing** and enable the previous two options (*Restore original screen* and *Restore OSD*). This way, both the Zebra pattern and the indicators are completely visible.
- Finally, there is a special Zebra mode that can be switched on by *RGB zebra*. This mode displays overexposed areas in different colors depending on which color channel is overexposed. For example, if the blue channel is overexposed, the Zebra pattern will be blue; if both the red and green channels are overexposed, the Zebra pattern will be yellow (red+green), and if all channels are overexposed, the Zebra pattern will be black. The drawback of this mode is that it cannot show underexposed areas.

4.3.5 High-speed photography

One amazing feature of the CHDK is the provision for ultra-short shutter speeds. Before we go into the details, let's take a look at general shutter technology.

Traditional cameras from the analog era use mainly two types of shutters. One is the focal plane shutter, which is also used in DSLRs. Here, two curtains move across the focal plane, and the light passes through a slit between the two curtains. The width of the slit determines the exposure time. By choosing a very narrow slit, a focal plane shutter can offer fast shutter speeds down to 1/8000 sec. The downside of this technique is that the slit needs time to travel across the focal plane. Fast-moving objects often appear stretched or contracted depending on the direction of the object's movement in relation to the slit's movement. Another problem with this type of shutter is flash photography. The entire sensor area must be uncovered by the curtains while the flash is emitting light in order to avoid uneven illumination. So shutter speeds cannot be very high when flash is used.

The other shutter type is the leaf shutter located in the camera lens. Here, several leaves quickly open the lens and then return to their original position. Because of this movement reversal, shutter speeds cannot be very short. For cameras with interchangeable lenses, each lens must be equipped with a shutter, and an auxiliary focal plane shutter is needed to cover the light-sensitive area during a lens change.

Compact digital cameras use a simple leaf shutter that is not very fast. Its main purpose is to shield the CCD sensor from light while the image is read. The exact shutter speed is determined by the point of time when the image data is read. So the CCD determines the shutter speed, not the mechanical shutter—and the CCD can be very fast, indeed. It also allows for using the flash with all shutter speeds. Most Canon compact cameras use CCD technology, though some newer models such as the SX1 use CMOS sensors.

CMOS sensors work differently. They do not need a mechanical shutter but work similarly to a focal plane shutter. Only a range of rows on the sensor—comparable to the slit between the curtains of a focal plane shutter—is light-sensitive at a given time. This range travels across the sensor area just like the curtain slit travels across the focal plane. Therefore, CMOS sensors tend to distort fast objects similarly to the traditional focal plane shutter. They also need slower synchronization speeds when using flash.

The CHDK can take control of the electronic shutter and offer shutter speeds down to 1/100,000 sec. That is more than the cameras can physically do, but in some cases real shutter speeds down to 1/40,000 sec. have been reported. A list of findings for different camera models is available at http://chdk.wikia.com/wiki/CameraFeatures. (Some of the top-range compact cameras such as the G-series cannot benefit from the CHDK in this respect because they use a mechanical shutter to control exposure time.)

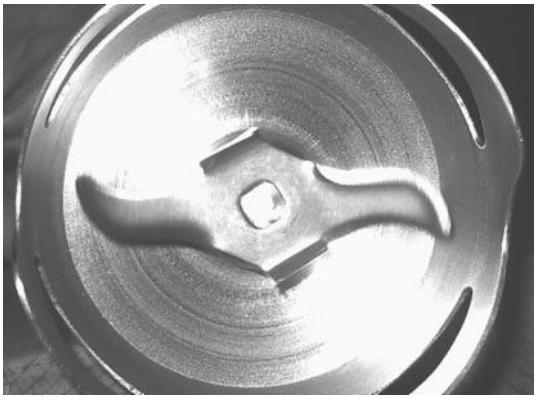


Figure 4-23
The blade of a power hand blender rotating at 15,000 rpm. The camera was a Canon Digital Elph SD1100 IS. In the CHDK Overrides, the shutter speed was set to 1/40,000 sec. and the ISO value to 1000. Flash was used for illumination. By measuring the extent of the small motion streaks on the blade, we could determine a shutter speed of 1/25,000 sec. The EXIF data, however, show a shutter speed of 1/1500 sec—the official maximum shutter speed.

When setting a fast *Tv* value in **ALT** > **MENU** > *Extra Photo Operations* > *Override Shutter Speed,* the aperture (if the camera has a diaphragm) and/or the ISO value must be adapted to achieve a correct exposure. Cameras with a

diaphragm usually offer a shutter priority automatic mode (also called *Tv* mode). On cameras without a *Tv* mode, you must set aperture and/or ISO manually, or use the script presented in section 5.7.4 (see also section 4.3.1).

4.3.6 Night photography

In the past, night photography has not been easy with digital cameras. At high ISO values, the noise in the image goes up. Shooting at low ISO values is not always possible—lack of a tripod may be just one reason. And at long exposure times, the camera's sensor tends to produce artifacts: hot pixels that show up as tiny bright dots. Hot pixels are the main reason why manufacturers limit the maximum exposure time.

However, inventive photographers have found a way around this limitation. The basic idea is: shoot at a higher ISO speed, but don't just take a single shot of your subject. Instead, shoot a whole series of images, each exposed in the same way. Afterwards, these images can be overlaid and averaged. This will cancel out some of the noise (see below), and the image will appear as if shot at a lower ISO speed. It is even possible to shoot such a series hand-held. PC-based programs like *PhotoAcute* can automatically register the images with each other and construct an almost noise-free composite image.

But what if you want to visualize motion (cars on the highway, star tracks, etc.)? In these cases you need long to very long exposure times. Without the CHDK, you are lost. But with the CHDK, your options for night photography are extended in several ways:

- *Shooting RAW*. RAW images can be treated with powerful PC-based noise reduction programs that outperform the in-camera noise reduction applied to JPEG files. These programs usually perform better with RAW sensor data. Your best option is the *DNG format:* here, bad pixels (as predetermined by the manufacturer) can be subtracted from the image (section 4.5.2). In addition, RAW images can capture the high contrast typical of night exposures better than JPEG images.
- *Higher ISO values* (section 4.3.1). This results in more noise, of course; but you can take several hand-held shots and average them later. Figure 4-24 shows that it becomes possible to obtain low-grain images of night scenes even without a tripod when using this technique.
- *Merging several RAW images* (section 4.5.6). This can be an option when you take a series of photos with your camera mounted on a tripod. This feature cannot register the images with each other, but it saves post-processing on the PC.

- *Longer exposure times* (section 4.3.1). How long depends on the camera model[4], but the minimum is 64 seconds. Many cameras allow for up to 2147 seconds.
- *Dark Frame Subtraction* (DFS) (section 4.5.2). DFS can be useful when you're shooting with long exposure times at low ISO speeds. In such conditions, it will effectively remove all hot pixels. When used with high ISO values, however, it can make things worse (see below). The CHDK allows you to switch DFS on or off, or to use the **Auto** setting when DFS will only be used for exposure times less than or equal to 1.3 seconds.

ABOUT NOISE

Each image contains noise—dominantly sensor noise (mostly in the blue channel) and amplifier noise. Small sensors cannot catch as much light as large sensors and thus require more amplification, resulting in a lower signal-to-noise ratio. The higher the amplification (and the ISO speed), the higher the noise.

Noise is a random phenomenon. Therefore, subtracting two independent noise sources from each other does not cancel out the noise. In this case, subtraction actually has the same effect as adding both noise signals. This is the reason why *Dark Frame Subtraction* does not cancel out stochastic noise but rather increases it. Where DFS helps is with *Hot Pixels* because they are not random.

Noise is reduced when averaging several images. Again, the reason is the stochastic character of noise. Adding the signals from n images results in an image signal of n-times the strength, but the strength of the noise signal multiplies only by the square-root of n. So, the image-to-noise ratio improves by the square-root of n. For example, when shooting a series of 16 shots and averaging them, you end up with an image where the signal-to-noise ratio is improved by a factor of $\sqrt{16} = 4$.

4 See http://chdk.wikia.com/wiki/CameraFeatures.

Figure 4-24
This detail of a night shot shows massive graininess in the top image. This was caused by setting the sensor speed to ISO 6400 (market) on a Canon Digital Elph 1100 IS (Ixus 80 IS). Exposure time was 1/20 sec, aperture f/2.8 at 6.2 mm (equiv. 38 mm). The image was shot hand-held as a DNG image with Bad Pixel Removal. The image at the bottom, in contrast, was composed of the image shown at the top and seven more similar images, all shot hand-held. Registration and averaging was performed with *PhotoAcute Studio*. No other noise reduction techniques were applied, so there is still room for further improvement.

4.3.7 Flash

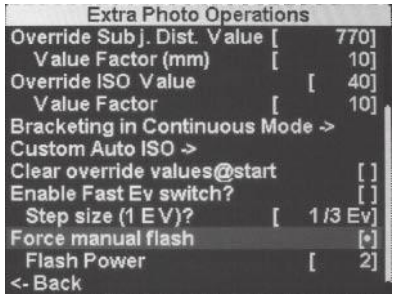


Figure 4-25
The commands for controlling the manual flash are reached through the menu *Extra Photo Operations*. You'll find them at the end of the menu.

Most Canon compact cameras have rather limited flash functionality. My SD1100, for example, supports the following modes: *Automatic, Off,* and *On* (in manual mode). Red-eye correction and a red-eye lamp can be switched on, and in manual mode there is a *Slow Sync* option. However, there is no way to control the flash power. This is unfortunate; it's hard to use the flash as a fill light.

Once again, the CHDK comes to the rescue with an override. There is an option to enforce manual flash even when the camera is in automatic mode, and the flash power can be controlled in three steps: *weak* (0), *medium* (1), and *strong* (2). When using the flash as a fill light, the medium and weak settings can be especially useful. I certainly would like a more fine-grained flash control with the option of controlling the flash power in 1/3 f-stops; but, you can't have everything. The three flash power settings of the CHDK are better than nothing. The strong setting comes in handy

when you need lots of light: for example, when you want to shoot with an ultrashort exposure time (section 4.3.5).

By the way, if you enforce the manual flash but it doesn't work, don't blame the CHDK; you might just need to charge the battery.

4.3.8 Using curves

Custom Curves are another option in the CHDK to control the outcome of a shot. Curves are applied after an image has been taken; they don't influence exposure settings such as aperture or sensor speed. They simply modify the digital data delivered by the sensor before it's packed into a JPEG file.

This can make sense. The sensor delivers the image data with a finer dynamic granularity than a JPEG file is able to capture. Each pixel consists of 10 or more bits, allowing the capture of at least 1024 levels of brightness. A JPEG file only transports 8 bits per pixel. Due to the logarithmic scaling of pixel values in JPEG format, the JPEG file does a good job on shadows but not so good a job on highlights. Therefore, burned-out highlights are a common problem when shooting in JPEG.

One solution is to shoot RAW (section 4.5). A RAW file delivers RAW sensor data—the complete information without any loss. Using a RAW converter, you can properly compress the tonal range of the image to avoid drowned shadows and burned-out highlights.

Is there no solution for shooting JPEGs? In fact, the CHDK offers one. Curves can be used to do all kinds of tone mapping, including contrast compression. The result can be a JPEG with detail in both highlights and shadows. Because curves are applied when mapping the RAW data onto JPEG pixel values, they do not have any effect on RAW images. And because the CHDK can shoot RAW and JPEG simultaneously, you can have both a curve-modified JPEG and an untreated RAW file as a backup.

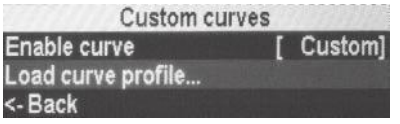


Figure 4-26
The *Custom curves submenu*

The *Enable curve* entry in the *Custom curves* submenu offers the following options:

- *None*. No curves are used for tone mapping.
- *Custom*. The pixel values are mapped under the control of a custom curve. Such custom curves can be created with the help of a curve editor (see below). The curve definition file must be placed into the folder `CHDK/CURVES/`. The curve can then be loaded under menu entry *Load curve profile...* .
- *+1EV*. A system-defined curve that increases shadow detail by one f-stop. Unlike the EV correction discussed in section 4.3.1, this option will not increase noise, open the aperture wider, or increase exposure time. It will, however, reduce contrast.

- *+2EV*. Similar, but increases shadow detail by two f-stops.
- *Auto DR*. This curve helps avoid burnt-out highlights in a controlled manner. Before taking an image, adjust the EV value (section 4.3.1) so that the highlights show good detail. Then take your shot. The luminance of the shadows will be raised by the curve while the highlights are compressed. In difficult lighting situations, this curve can be useful—but you should not expect wonders. Better results can be achieved by shooting RAW (section 4.5) and by adapting the tonal values manually on the PC, or, of course, by shooting an exposure series for creating an HDR image (section 4.6.2).

These three system curves are contained in file `CHDK/syscurves.CVF`. This file is loaded automatically.

Custom curves can be created with a PC-based curve editor running under Windows. It is available from the web page http://chdk.wikia.com/wiki/Software and from the book CD. To install the editor, simply copy it onto the desktop. To create a new curve, select a color channel or *All* in the *Edit* group, then start setting and dragging points with the mouse. You can save the curve by entering a name into the text field at the bottom, then click *Save Point*. This will create a `.CTC` file that is a plain text point list. For usage in your camera, you must compile the curve with *Create Curve*. This will produce two files: a `.CV` file and a `.CVF` file. Copy these files into the folder `CHDK/CURVES/` on your memory card.

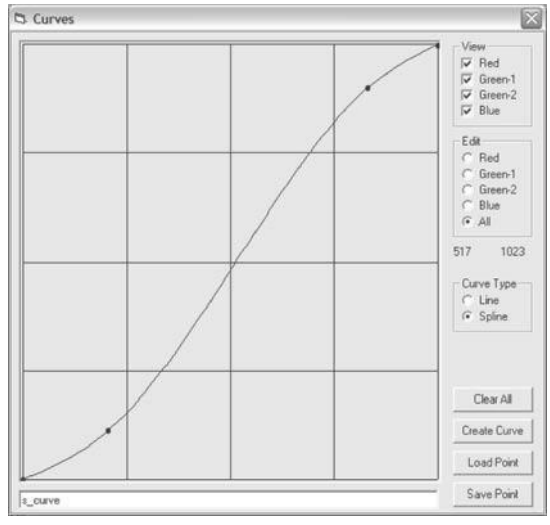


Figure 4-27
The curve editor in action. Here we have created a typical S-curve that compresses the tonal values in the shadows and highlights and boosts the midtones. The curve mimics the response curve of photographic emulsions.

CV-curves are plain RGB curves. CVF-curves, in contrast, introduce a color shift for emulating an application of the curve to the luminance channel of the YUV color model. Usually, this gives a better visual result in areas with strong brightness changes.

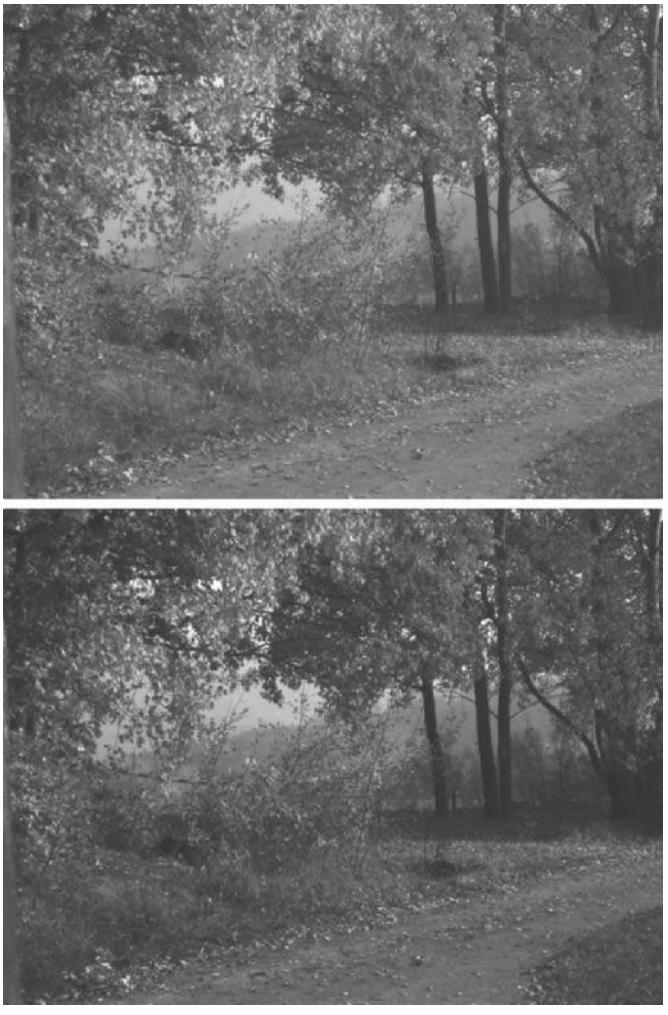


Figure 4-28
Application of the S-curve from Figure 4-27 to an autumn scene. The upper image was delivered as a DNG file, the lower image as a JPEG file. As you can see, the JPEG image has more contrast in the midtones.

You may wonder why the curve editor supports two green curves *(Green-1, Green-2)*. The reason is that the camera's sensor cells are divided into *Red, Green-1, Blue,* and *Green-2,* resulting in two green channels. Four sensor cells can be nicely arranged in a 2x2 pattern. Green was duplicated because the human eye is especially responsive to green colors.

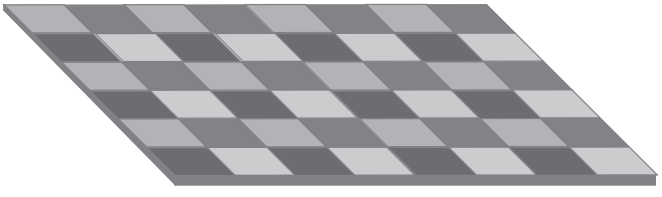


Figure 4-29
The photocells of the camera sensor are not sensitive to color. The Bayer filter—placed in front of the sensor—assigns a specific color to each cell (Red, Green-1, Blue, or Green-2). When the RAW sensor data is developed by software—either in-camera or on a PC—the full color pixel values are interpolated by fetching the missing colors from the neighbor cells.

Custom curves are, of course, not restricted to compression of the dynamic range. They can also be used for creative purposes. When you assign different-shaped curves to different color channels, you may get interesting color shifts. Curves with hills and valleys result in an effect known as the *Sabbatier* effect.

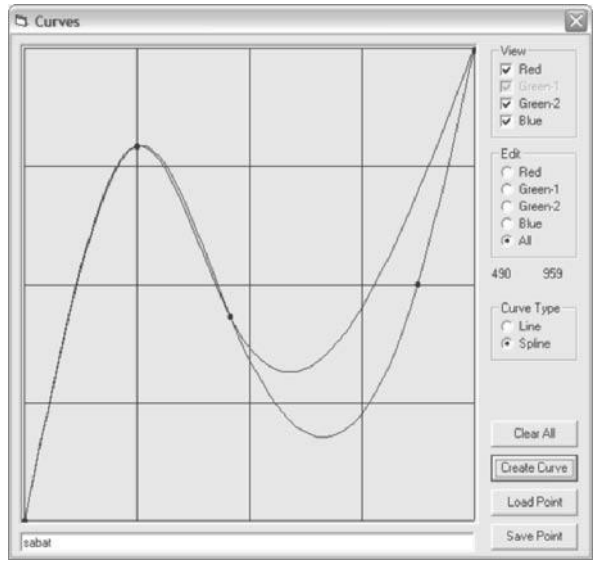


Figure 4-30
Typical curve for achieving a Sabbattier effect. The different curves for the single color channels will cause color shifts in midtones and highlights.

When doing such work, you should shoot RAW in any case. The curves are only applied to the JPEG output[5]. If you are not happy with the outcome, you still have the original and unmodified image in the RAW file. And by redeveloping the RAW file (section 4.5.5), you can try out another curve.

As a matter of fact, all of these modifications using curves would also be possible in a post-processing stage—at least when you opt to shoot RAW. Image editing programs such as *Photoshop* or *Paintshop Pro* allow for such modifications of the tonal range. However, using curves in the camera may help you to realize your visualization of the scene earlier and to redo a shot when the outcome does not match your expectations.

5 They also affect the preview images embedded into DNG files because the previews are derived from the JPEG image.

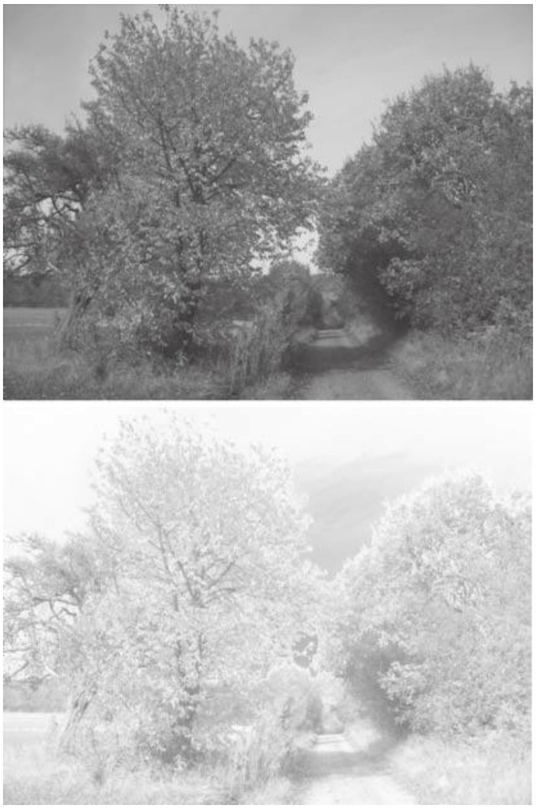


Figure 4-31
Dream landscape obtained with the curve shown in Figure 4-30. See the tonal inversion in the central sky area.

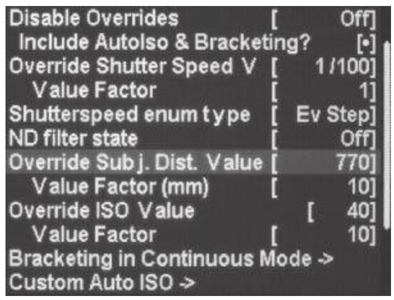


Figure 4-32
The entry *Override Subj. Dist. Value* in the submenu *Extra Photo Operations* can be used for manual focusing. The override can be switched off via the subentry *Value Factor*. The labeling of this subentry is somewhat misleading. Unlike its equally named siblings, this value factor does **NOT** define a value factor by which the main entry is multiplied. Instead, it defines a **step width** for modifying the main entry. So, the subject distance set here is 770 mm (77 cm) and not 7700 mm as you might expect.

4.4 Focus

The focal system of Canon compact cameras is dominantly based on an autofocus system. Only some cameras in the higher price range allow manual focusing. The autofocus system is augmented by special focus modes such as *Macro* or *Infinity,* which can be set manually. Over the years, the focus system has become more and more advanced. One of the more recent improvements is face detection, allowing the camera to focus precisely on the faces in a scene.

Like any of the focus systems in other compact cameras, the Canon AF system is also contrast-based. When focusing, the camera moves the lens back and forth until it finds a position where the contrast in selected image areas is maximized. This takes some time; the phase detection-based focus systems in DSLR cameras are much faster.

In the area of focusing, the CHDK also adds functionality. An explicit subject distance can be dialed in precisely to the millimeter. This overrides the autofocus system as well as selected focus modes such as *Macro* or *Infinity*. The value range is between 0 and 65535 mm, with 65535 representing infinity. The easiest way to dial in infinity is to start with zero and

press the LEFT button once. When using focus overrides, cameras featuring a manual focusing mode must be switched into that mode.

Like any other override, the subject distance setting can also be controlled via a script that allows you to do amazing things, such as automated focus stacking. We will discuss these techniques in section 5.7.2.

4.5 Shooting RAW

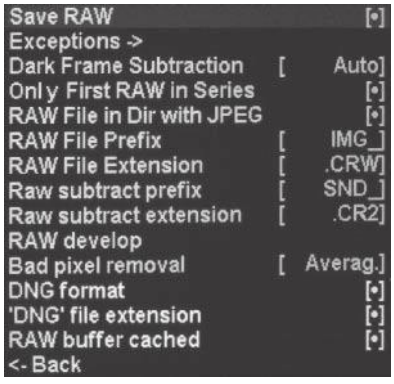


Figure 4-33
The *RAW* menu adds functionality for outputting RAW files. The first menu item, *Save RAW,* allows you to toggle quickly between RAW and non-RAW shooting.

The RAW image format has so many advantages for the semi-professional photographer that all cameras should offer it. However, this is not the case. In particular, many compact cameras do not offer this option. To be able to produce RAW images with consumer cameras was the main rationale behind the development of the CHDK.

4.5.1 Basics

RAW images are nothing more than the unmodified sensor data produced by the camera. JPEG images, in contrast, are the result of a development process: the color temperature is estimated and applied, the image is sharpened, noise is removed, and the contrast is compressed. Scenes with stark contrast usually have some highlights clipped. So, the camera has already made some decisions for you.

A RAW image, on the other hand, allows postponing these decisions to a later time. You don't have to decide on processing parameters before taking a picture. Instead, you can make these decisions at the post-processing stage when the RAW image is developed on a PC (or even in the camera). It is possible to play around and try different options in regard to exposure, white balance, sharpening, and noise suppression.

Another big advantage of the RAW format is that you can decide which color space to use for the image during the development process. When shooting JPEG, the camera makes this selection for you—and most compact cameras won't allow anything but the small sRGB color space. The RAW format does not cast the RAW sensor data into any preset color space. This decision is made later when the images are developed in a RAW converter on a PC. Most PC-based RAW converters allow you to use the wider AdobeRGB color space. Some even allow for a larger variety of color spaces, such as *ProPhotoRGB* or *eciRGBv2*.

sRGB VS. AdobeRGB

The *sRGB (standard RGB)* color space was defined by Microsoft and Hewlett-Packard. It is used by basically all consumer devices such as monitors, printers, scanners, and LCD projectors. It is also used as the standard color space in the Windows operating system. Because this color space is fairly narrow, the devices can be simple and cheap. The downside is that the *sRGB* color space does not cover some colors used in the printing process—so prints are not as colorful as they could be.

The *AdobeRGB* color space, in contrast, was designed by Adobe to meet the demands of the printing press. It is larger than the *sRGB* color space. However, the problem with this color space is that *AdobeRGB* images cannot be properly viewed on standard monitors. A *Wide Gamut Monitor* is required to view the full color range.

So, if your intention is to produce images for the web, by all means go for *sRGB* because the majority of your audience will not be able to display the full range of *AdobeRGB*. But if the printed image is your main concern, go for *AdobeRGB* and consider a *Wide Gamut Monitor* for your next monitor.

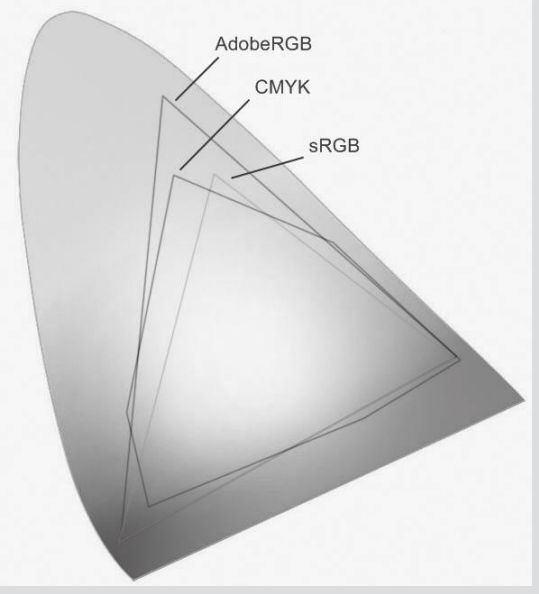


Figure 4-34
AdobeRGB covers the sRGB color space as well as most of the CMYK color space used for printing. Modern camera sensors capture an even larger color space.

Also, a PC-based RAW converter usually does a better job than the processor in the camera. This is particularly true for the small consumer cameras for which the CHDK was developed. The small processors in those cameras cannot run sophisticated interpolation algorithms in the short time after a shot. In addition, when converting RAW files on the PC, the output of the

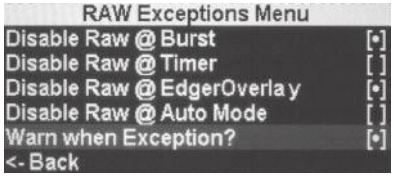


Figure 4-35
The *RAW Exceptions Menu*. Not every situation requires RAW output, especially if you need faster camera operation and more capacity for images on your memory card.

RAW processor can be a 16-bit lossless TIFF file instead of a lossy JPEG file that is restricted to 8-bit color depth.

Of course, there are good reasons for shooting JPEG, too. Because JPEG files are much smaller than RAW files, they can be stored faster on the memory card, resulting in faster operation when shooting a series—and, of course, in higher capacity on your card. Also, you may not always want to go through the time-consuming process of developing RAW files.

CHDK gives you the best of both worlds. When shooting RAW, the CHDK also stores a JPEG version of the same image for preview. If you are happy with the JPEG image, you can throw away the RAW file. If you think the image requires some tweaking, keep the RAW file. CHDK even allows you to exclude some situations from RAW shooting (**ALT** > **MENU** > *RAW Parameters* > *Exceptions*):

- **Disable RAW @ Burst**. Burst mode is faster when not shooting RAW. There is also an option in the main *RAW* menu (Figure 4-33) to shoot the very first shot of a series as RAW but the following shots as JPEGs.
- **Disable RAW @ Timer**. You may not want to shoot your self-portraits in RAW mode.
- **Disable RAW @ EdgerOverlay**. *Edge Overlay* is typically used with shooting stereo and panoramic images. As many panorama stitcher and stereo programs don't accept RAW files, JPEG is the preferred option here.
- **Disable RAW @ Auto Mode**. You probably have your camera switched to *Auto Mode* for casual shots. You would hardly need RAW images here. For more serious shots where the RAW format makes sense, you would switch your camera to *Manual Mode* because it gives you more control.
- **Warn when Exception?** Optionally, a warning can be displayed when one of the above situations applies.

4.5.2 DNG

The RAW format produced by the CHDK has some restrictions. First, CHDK's RAW images are not equipped with EXIF data. Second, the RAW format produced by the CHDK is not understood by all RAW converters. Some simply refuse to open the image. A good choice is the program *RawTherapee*, which is available for the main PC platforms *(Windows, Linux, Mac)* on a donation basis. It's a bit slow but delivers excellent results. Also, the free command line tool *dcraw* converts CHDK RAW files with excellent quality.

Newer versions of the CHDK, however, can produce RAW images that are already converted to the DNG format. Adobe Systems introduced DNG in order to bring different RAW formats (each manufacturer has at least one) under one umbrella. DNG files still contain RAW data but in a manufacturer-independent, Adobe-specific way. Some premium cameras such

as Hasselblad, Leica, and Pentax use DNG as their native RAW format. With the CHDK, you can now join this exclusive club and produce DNG files directly from your camera.

Using DNG has a few advantages over the native RAW format:

- You can choose among a large variety of image editors and *RAW developers* that accept DNG files as input.
- DNG files produced by the CHDK do contain *EXIF data*.
- *Bad pixels* are optionally corrected. There are three options in the menu entry ALT > MENU > *Raw Parameters* > *Bad pixel removal:*
  - **Off**. No bad pixel removal.
  - **Averag**. Replaces the bad pixel with an average from its neighbors.
  - **RAWconv**. The repair job is left to the PC-based RAW converter. The list of bad pixels is included in the DNG file.

Because bad pixels are optionally considered during the in-camera conversion to DNG, a file `badpixel.bin` is required by the CHDK. Only then will the DNG menu items work. So, before using DNG, you must execute the script `badpixel.lua`. This script, which is included in the CHDK distribution, will read the list of bad pixels (as determined by the manufacturer) from the camera's firmware. To run the script, switch the camera to *Recording Mode* and invoke:

1. Alt > FUNC/SET > *Load Script from File...* > *TEST* > *BADPIXEL.LUA*
2. Press the shutter button. The display will go dark, and the camera will take **two** pictures. After 20–30 seconds, the display will reappear and the camera will wait for input. If the bad pixel test was not successful, the script will ask you to run the test again.
3. Press FUNC/SET to save the file.
4. Press ALT to switch back to normal mode.
5. Now you can go ahead and switch to DNG (ALT > MENU > *RAW Parameters* > *DNG Format*). Or, you can use the built-in file browser to check that the file `BADPIXEL.BIN` has been created in the folder `CHDK/` (ALT > MENU > *Miscellaneous Stuff* > *File Browser*).

When working with DNG files, you should also enable the menu entry *RAW buffer cached*. The built-in DNG converter will then read the RAW data directly from memory and not from the card, resulting in faster operation. However, if you frequently run into memory problems, disable this option.

DARK FRAME SUBTRACTION AND BAD PIXEL REMOVAL

Do not mistake *Bad Pixel Removal* for *Dark Frame Subtraction*. Both have their pros and cons:

Dark Frame Subtraction is a common technique for removing unwanted artifacts from an image. When an image is taken, the camera will make two exposures: one with an open lens, the other with a darkened lens. The second exposure will only show noise and hot pixels. Subtracting the second image from the first will remove some of these artifacts. Dark frame subtraction is usually used for long night exposures where the white, hot pixels can be very disturbing. As a noise reduction technique, it is of limited value; subtracting noise from noise results in even more noise because noise is a random phenomenon. Therefore, this technique should only be used in combination with rather low ISO settings. For better noise reduction strategies, see sections 4.3.6 and 4.5.5. With the CHDK, you have the option of switching *Dark Frame Subtraction* on or off, or of using it only for exposure times longer than one second *(Auto)*.

Bad Pixel Removal works quite differently and only in connection with DNG. A *bad pixel map* is created in advance. In section 4.5.2, we already discussed how such a map is created. When running, the script `badpixel.lua` grabs a bad pixel map from the camera (normally used for the creation of JPEG images) and stores it onto the card. The map itself is created when the camera is manufactured and is part of the firmware. Therefore, and in contrast to *Dark Frame Subtraction, Bad Pixel Removal* catches not only hot pixels but also dark, dead pixels.

Because *Bad Pixel Removal* only replaces single pixels, it does not worsen the overall signal-to-noise ratio. The catch is that the bad pixel map is static and doesn't reflect the fact that bad pixels increase with both exposure time and the lifetime of the camera.

4.5.3 Other RAW parameters

RAW files saved by the CHDK are not equipped with EXIF data (DNG files are). This sounds like a drawback because EXIF data are very useful for processing and archiving images. Vital information such as exposure time, aperture, ISO speed, focal length, distance, etc., would be missing without the EXIF data. But because the CHDK always stores both a RAW file and a JPEG file (which does contain EXIF data), the missing EXIF data in the RAW file can be easily restored from the JPEG file.

Prefixes and Suffixes

Tools such as *DNG4PS-2* (section 4.5.4) and *ZoRa Photo Director* can automatically pick the EXIF data from a corresponding JPEG file when processing RAW images—provided that the image name starts with the prefix `IMG_` and that both the JPEG and RAW files are stored in the same folder. You should enable the option *RAW File in Dir with JPEG* and set the parameter *RAW File Prefix* to `IMG_` in order to make use of these tools.

The CHDK allows you to select different file name extensions for RAW files. The purpose of this is to trick image uploaders that are not designed to upload RAW images into uploading RAW files. However, if your image uploader can handle the CRW file extension, or if you work with a card reader, you should definitely use the CRW extension.

Similarly, you should use the extension DNG when you use DNG as your output format. Enable the option *'DNG' file extension*.

4.5.4 Processing RAW images

Many photo editors such as *Photoshop, Paint Shop Pro, Picture Window,* and so on can import RAW images. And then there are the specialized, workflow-oriented RAW developers such as *Aperture, Lightroom, CaptureOne, Helicon Filter, BreezeBrowser, SilkyPix, DxO, RawTherapee,* and others. These programs have highly specialized tools for developing RAW images. Many lens and sensor imperfections—such as chromatic aberration, pincushion distortion, vignetting, sensor noise, and dead or hot pixels—can be corrected. You will be surprised by the quality you can get from the lens of your compact camera.

Unfortunately, not all of the above programs are happy with the RAW format produced by the CHDK. One notable exception is the free RAW developer *RawTherapee,* which does an excellent job of converting CRW files into TIFFs or JPEGs.

Your other option is to use the DNG format as RAW format (section 4.5.2). Alternatively, you can do the conversion from RAW to DNG on a PC with the help of the free program *DNG4PS-2*.

You may very well ask, which is better: to conveniently create the DNG file in-camera, or to shoot in native RAW format and let *DNG4PS-2* convert the images to DNG? The results are actually quite similar. This makes sense, since RAW-to-DNG conversion never modifies the sensor data but only repackages it. The only exceptions are the bad pixels. When creating the DNG file in camera, bad pixels can be removed (section 4.5.2). This is not the case when you create a CRW file and convert it to DNG with DNG4PS.

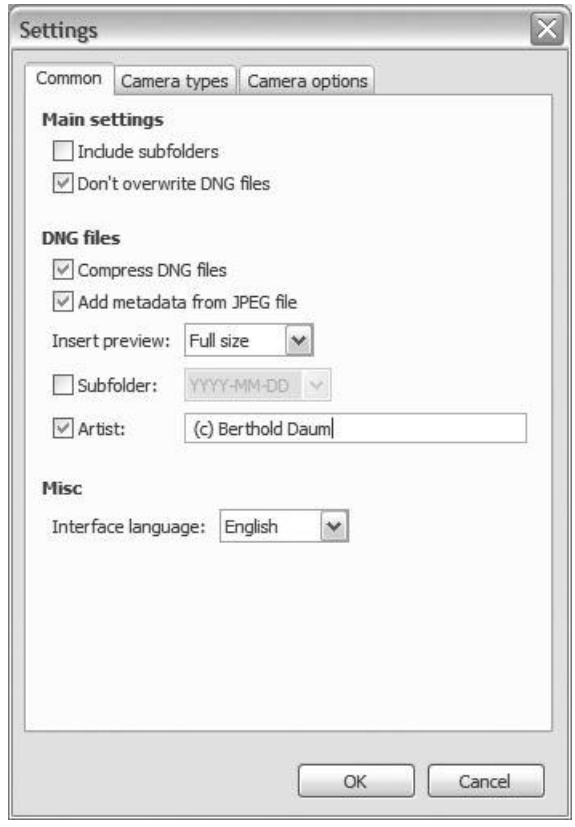


Figure 4-36
The *Settings* dialog of DNG4PS. The program is able to furnish the resulting DNG files with the EXIF data found in the corresponding JPEG files. You must specify your camera under the tabs *Camera types* and *Camera options*.

On the other hand, the DNG files produced by DNG4PS are about 30 percent smaller. DNG4PS applies some lossless compression, which the CHDK does not do—for speed reasons, obviously. But if disk space is an issue, you could always feed the resulting DNG files into the *Adobe DNG Converter* and compress them even further! That is actually all the *Adobe DNG Converter* can do for you—in particular, it does not understand the RAW format CRW produced by the CHDK.

So, yes, I prefer the in-camera creation of DNG files.

4.5.5 In-camera RAW processing

You don't necessarily need a PC to process RAW files. With the CHDK it is possible to develop RAW files within the camera. The selected RAW file will be converted into a JPEG file.

Why should you do this? The advantage is that you can apply some processing parameters after a shot has been taken, for example *Color Accent, Color Swap,* one of the choices in *MyColors,* or a different *White Balance* setting. You can even create different versions from the same RAW

original. And because custom curves (section 4.3.8) are considered during RAW development, you can create different variations of the image using different curves.

How do you do this? It's very simple:

1. Create your original file as a true RAW file (not as a DNG file).
2. Invoke the menu entry **ALT** > **MENU** > *RAW Parameters* > *RAW Develop*. This will bring up the file browser (section 4.10.1).
3. Navigate to the image folder (e.g., `CANON101/`), find the image (using date and time for orientation), and press **FUNC/SET**.
4. The camera will now ask you to switch to *Recording Mode*. To do so, press **ALT** again. If the camera is in *Playback Mode,* switch to *Recording Mode*.
5. Set your processing parameters (see above). See the manufacturer's manual for instructions.
6. Press the shutter button. Instead of taking a shot, the camera will read the RAW file and produce a JPEG file using the current camera settings.

4.5.6 More RAW processing

The possibilities don't stop with the RAW processing discussed in section 4.5.5. The context menu of the file browser (see section 4.10.1) opens a whole new world of addititional RAW processing options. You can reach the file browser through **ALT** > **MENU** > *Miscellaneous Stuff* > *File Browser* or **ALT** > **MENU** > *RAW Parameters* > *RAW Develop*.

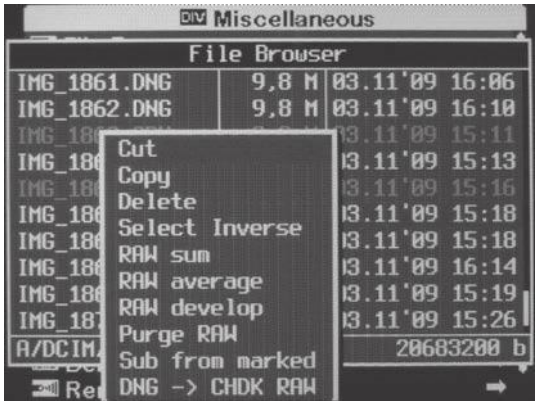


Figure 4-37
In the file browser, you can mark files by pressing the RIGHT button on each file that you want to select. Pressing the RIGHT button on a marked file removes the mark. Pressing the LEFT button pops up the context menu. Pressing LEFT again closes the context menu.

Besides the traditional file browser operations such as *Cut, Copy, Delete, Select Inverse* (section 4.10.1), there are some interesting RAW processing functions. Again, they only apply to true RAW files, not to DNG files.

- **RAW sum**. This function adds the pixel values of all marked RAW files. The effect is similar to taking an exposure with a longer exposure time. For example, if you add 10 images, each exposed at 15 seconds, the resulting image will almost look like an image taken with an exposure time of 150 sec. This operation is very similar to the double exposures taken with analog cameras.
  The resulting image gets the same name as the last of the marked images, but the file name extension is set to `.WAV` in order to identify the file as the result of the RAW merge operation.
- **RAW average**. A similar operation, but it computes the average of the pixel values instead of their sum. Thus, 10 images exposed at 15 seconds will produce an image with the exposure time of 15 seconds. The main reason for using this operation is to reduce noise. Averaging images is the most effective method of noise reduction without losing detail. Make sure that the single images are perfectly aligned and that there are no moving objects. Mounting the camera on a sturdy tripod and using the series function (section 4.6.2), an intervalometer script (section 5.7.1), or a remote control (section 4.9) is a must. If you don't have a tripod at hand, see section 4.3.6.
- **RAW develop**. This function performs the same as the one discussed in section 4.5.5 but applies RAW development to all marked images.
- **Purge RAW**. This function purges all orphaned RAW files (but not DNG files) that don't have a corresponding JPEG file from a selected folder. When applied inside a folder, it purges only orphaned RAW files that are not marked. Therefore, marking a file can protect it from the *Purge* operation. Typically, the *Purge* operation is used after you have deleted some images with the native camera function. The native function only deletes JPEG images, not the corresponding RAW files. So, purging can reclaim some memory. **Warning:** Purging cannot be undone. It is safest to do a backup of the card before applying the *Purge* function.
- **Sub from marked**. This function subtracts the currently highlighted image from all marked images. The resulting files get names composed from the original file names, the *Raw subtract prefix,* and the *Raw subtract extension*. Typically, image subtraction is used to detect movements in a scene (for example, astronomers find new supernovas by subtracting images). Again, use a sturdy tripod and a self-timer or remote control (section 4.9) to trigger the shutter. If you unintentionally introduce some shift between the images, *relief images* are more or less the result.
- **DNG > CHDK RAW**. The operations above all require true RAW files. But if your shots are DNG files, all is not lost. Using this function, you can convert marked DNG files back to RAW.

The functions *Average, Sum,* and *Develop* can also be invoked programmatically via a *Lua* script (section 5.5.9), which allows automating such superimposition tasks completely.

4.6 Bracketing

Bracketing is a camera function that produces a series of photos with varying settings. Traditionally, bracketing was used to obtain an image with perfect exposure. First, an image with the measured exposure was made, then an image half an f-stop overexposed, then an image half an f-stop underexposed, then the same with a full f-stop. Even analog cameras had the ability to perform bracketing in an automated fashion—which wasted a lot of film.

Today, with digital imaging, bracketing can still be used to obtain a perfectly exposed picture. It also has found new applications *[Gulbins2009]*. Combining the different pictures of a bracketing series into a single image has become quite easy, and new algorithms for image processing provide results that analog photographers can only dream of.

The traditional exposure series can now be used to combine the differently exposed images into a single image with a wider dynamic range. This type of photography—called *High Dynamic Range* (HDR) photography (section 4.6.2)—has become fairly popular among serious amateur photographers. The results often resemble a painting rather than a photograph. During post-processing, you have many options for compressing the huge dynamic range into a smaller and printable (or viewable) dynamic range. It is certainly a matter of personal preference and taste as to how the results will look.

Exposure values aren't the only items that can be bracketed; the same is possible with focal distance. Starting from a measured or manually set distance, the camera can make exposures with longer distances and shorter distances (section 4.6.3). Later, during post-processing on a PC, the images can be combined with the help of a special focus stacker. These programs select the sharpest parts of each image and combine them into a single image. The results are images with an extended *Depth of Field* (DOF). This technique is typically used in macro photography where the depth of field is very small.

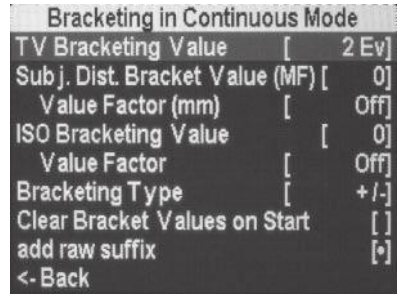


Figure 4-38
The *Bracketing* submenu is reached via ALT > MENU > *Extra Photo Operations* > *Bracketing in Continuous Mode*. Here we see a typical configuration for shooting HDR images.

4.6.1 General bracketing notes

Setting up the camera for an unlimited series of bracketing shots is very easy under the CHDK. Just go to the *Bracketing* submenu and set the values for the entries found there:

- *TV Bracketing Value*. This entry is used for modifying the exposure time during a series. A value of 2 results in an exposure time multiplication factor of $2^2 = 4$. So, if you start with 1/100 sec, you will get a shutter speed series of 1/100, 1/25, 1/400, 1/6, 1/1600, and so on. This type of bracketing is typically used for HDR work (section 4.6.2). It can also be used for getting the perfect exposure when AV bracketing is not available.
- *AV Bracketing Value*. This entry allows you to modify the aperture for the individual shots in a series. For example, if you set the value to one f-stop and the aperture to 5.6, you will end up with a series shot at apertures 5.6, 4.0, 8.0, 2.8, 11, etc. AV bracketing is not suited for HDR work because the depth of field changes with each aperture change; but it's a good option for getting a perfectly exposed single image. This menu entry is only available on cameras with a diaphragm.
- *Subj.Dist.Bracket Value*. This entry is used for focus stacking, which is discussed in the next section.
- *ISO Bracketing Value*. This entry allows you to modify the ISO value. The subentry *Value Factor* specifies a factor that is multiplied with the *ISO Bracketing Value*—just to save you a few hundred keystrokes. The result is added and subtracted from the original ISO value. For example, if you start with an ISO value of 100, an *ISO Bracketing Value* of 2, and a *Value Factor* of 10, you will end up with a series of 100, 120, 80, 140, 60, etc. ISO bracketing is well suited for classical exposure bracketing when you need the perfectly exposed single image, especially if AV bracketing is not available. It is not as well suited for HDR work (see above) because the signal-to-noise ratio changes with each ISO setting.
- *Bracketing Type*. There are three bracketing types. Choosing "+/–" will result in a series with alternating over- and underexposed images, typically used for finding the perfect exposure and for HDR work. The option "–" will result in a series of increasingly underexposed images. Finally, the option "+" will result in a series of increasingly overexposed images. The last two options are often used for focus stacking.
- *Clear Bracket Value on Start*. If this option is enabled, the bracketing values will be reset to default when the camera is powered up.
- *Add RAW Suffix*. When this option is enabled and you shoot RAW images, a suffix is added to the names of the resulting RAW files, for example, `IMG_2041_01.DNG`, `IMG_2041_02.DNG`, `IMG_2041_03.DNG`. This allows you to easily identify the bracketing series later. However, in the CHDK file browser of DryOS cameras (section 2.2), the files will show up as `IMG_20~1.DNG`, `IMG_20~2.DNG`, `IMG_20~3.DNG`, because the file browser is limited to 8.3[6] filenames on this platform.

6 At most, eight characters in front of the dot, three behind it.

To enable bracketing, you must also set *Disable Overrides* in the submenu *Extra Photo Operations* to **Off** (section 4.3.1) if you enabled the option *Include AutoISO & Bracketing* in the same submenu. Then switch the camera to *Continuous* mode. On cameras that support multiple shots taken with the *Custom Timer, Custom Timer* is another possible option. This would allow you to predefine the number of shots to be taken.

4.6.2 HDR and tone mapping

High Dynamic Range (HDR) and *Dynamic Range Increase (DRI) Photography* have become quite popular among digital photographers. Both techniques are used for recording contrast ranges that are beyond the maximum contrast range of a state-of-art camera sensor *[Howard2008][Bloch2007]*.

Photographic scenes—especially those in full sunlight—can have a dynamic range of 16 to 25 *Light Values* (LV) or f-stops. Even the human eye cannot capture such extreme contrast (it is limited to approximately 14 LV) but must adapt itself when looking at dark or bright areas. The eye can do this because it scans only a narrow area of a scene. The signals produced by the eye are then processed by the brain, which composes them into a whole scene again.

The camera does not have this ability. It must record an entire scene in one step and therefore should be able to capture all the contrast in a scene. That, of course, is not yet possible. Modern camera sensors can record a dynamic range of 11–13 LV, a theoretical value that is lowered by noise and other imperfections. In real-world applications, a camera sensor can capture a contrast range of 8–10 LV with good quality. When showing an image on an LCD screen, a dynamic range of 5–7 LV can be reproduced, and when printing an image on photo quality paper, 6 LV is a reasonable assumption.

Because of this situation, you have to decide on the brightness subrange that you wish to record—meaning that you must choose the correct exposure. Brightness values outside that range will record as totally black shadows and totally white highlights with no detail.

The contrast in the recorded image, which by now is 8–10 LV, must still be compressed in order to reproduce all details in the image on a print or on a screen. Typically, this is done in a photo editor such as *Photoshop* or *Paintshop Pro* by modifying the brightness curve. With the CHDK, however, it can also be done in-camera by using *Custom Curves* (section 4.3.8). This tone mapping requires a bit of skill from the photographer; a brightness curve that is too steep will result in contrasty images with blown-out highlights and dead shadows. A brightness curve that is too flat will result in a flat picture, and vice versa.

Enter the HDR method. The higher dynamic range is simply achieved by making several images of the same subject: an image with the correct exposure, an image two f-stops overexposed, and an image two f-stops underexposed. You can go even further and make additional images: +4 f-stops, –4 f-stops, +6 f-stops, and –6 f-stops. Given a camera sensor contrast range of 8 LV (which should be a reasonable value for a compact camera), we can capture a scene brightness range of 12, 16, or 20 LV with this method.

There are a few rules to follow when creating such a bracketing series:

- Zoom value, focus, aperture, white balance, and (if possible) sensor speed must not be changed because such changes cause image changes. All bracketing should be done by changing the shutter speed.
- The sensor speed should be set to a low value, such as ISO 50 or ISO 100, to avoid sensor noise.
- There should be no moving objects in the scene. Some HDR composers are able to remove moving objects (ghosts) from an image, but not always with good success. So beware of cars, moving people and animals, and wind (leaves and branches move). At night, even the stars and the moon can cause problems.
- The illumination of the scene should not change during a bracketing series. Otherwise, it will be difficult for the HDR composer to put the images together correctly.
- It is certainly an advantage to put the camera on a tripod, but modern HDR composers are able to register the single images with each other even if they are taken hand-held.

If all these conditions are met, you can set up your camera to shoot a bracketing series:

- Typically you would use a *TV Bracketing Value* of 2 and the *Bracketing Type* "+/–".
- If you want the best quality, shoot RAW images. If you do, make sure that the options **ALT** > **MENU** > *RAW Parameters* > *Only First RAW in Series* and **ALT** > **MENU** > *RAW Parameters* > *Exceptions* > *Disable Raw@Burst* are disabled.
- You now have the choice of setting the camera to *Continuous* mode, or of setting up the *Custom Timer* with 0 seconds delay and 3, 5, or even 7 shots. If you use *Continuous* mode, you need to count the pictures yourself. The advantage of this mode is that you can pause the series when somebody or someone moves into the scene. Just half-release the shutter button and press it fully when you want to continue. The *Custom Timer,* in contrast, does all counting for you—you just have to press the shutter button.

After you have taken the individual images, the image series must be post-processed and combined into one single image. If you produced RAW images, you should develop them with a RAW developer (section 4.5.4). Even if your HDR image composer is able to accept RAW files, a RAW developer will give you superior results because of the corrections that can be applied to lens and sensor imperfections. You should at least reduce the chromatic aberration and the noise and apply a first sharpening to the images. Start with the first image in the series—the "correctly" exposed image—and then apply exactly the same development parameters to all of the images in the series. Most RAW developers allow you to copy the development parameters from one image and apply them to others. Of course, your target file format would be TIFF and not JPEG because TIFF is lossless.

Next comes the step of composition. HDR and DRI composers know two techniques for composing HDR series:

- *Weighted average*. The pixel values of all images are added, but a weight is applied to each pixel value. Pixel values of a dark pixel in an overexposed image get a higher weight, as do pixel values of a bright pixel in an underexposed image.
- *Area oriented*. The picture is segmented into different brightness areas. Each area is treated differently and may get its pixels from a different source image. With this method, there is almost no blending of images. Therefore, it is well suited for images with moving objects in the scene. Ghosts are hardly possible because most pixels come from only one source image.

After composition, DRI and HDR go separate ways. DRI immediately applies tone mapping to reduce the tonal range of the image to a viewable and printable size. It therefore can use standard file formats such as TIFF or JPEG for the output.

HDR, in contrast, preserves the original dynamic range of the scene. Therefore, it must use specialized file formats such as *OpenEXR, Radiance HDR,* or *32-Bit-TIFF*. Traditional file formats with 8 or 16 bits per pixel and channel cannot capture the high dynamic range of an HDR image. Once you save the composed image into an HDR-specific file format, you can open the file later and decide how to map the dynamic range of the image onto a smaller printable or viewable dynamic range.

This brings up the technique of tone mapping: reducing the huge dynamic range of the composite image into something smaller. Again, there are different methods to achieve this:

- *Global tone mapping* applies the same formula (in most cases, a logarithmic formula) onto each pixel. This is fast but can easily result in flat images. This technique is suitable for images with low or medium contrast.
- *Local tone mapping* takes the neighboring pixels into account. Brightness differences between neighboring pixels are weighted higher than

brightness differences between distant pixels. This technique results in good local contrast and is well suited for images with high contrast. The images usually look crisper than the ones created with global tone mapping because local structures are emphasized. When overdone, however, artifacts such as halos can become visible.

Most of the established image editors such as *Photoshop, Paintshop Pro,* and *Picture Window* support the creation of HDR and DRI images. For serious HDR work, however, we recommend a specialized HDR composer such as the free *Picturenaut (Windows),* or *HDR PhotoStudio, Hydra (Mac only), Photomatix Pro, FDRTools Advanced,* or *Dynamic Photo HDR*. For an example, please see Figure 6-1.

4.6.3 Focus stacking

Depth of Field (DOF) is often a problem in photography. Only a distance range within the image is sharp, while objects outside that area are blurred. In particular, telephoto shots, macro work, and tabletop photography are hampered by this problem. The traditional remedy is to stop down (or, in the case of view camera and tabletop photography, to tilt the camera's back). Remember *Group f/64—Ansel Adams* and friends? The name was derived from the fact that you had to stop down the large view cameras to f/64 to obtain sharpness from near to far. Large cameras with long focal lengths are, in fact, more affected by this problem than small cameras with short focal lengths.

Nevertheless, small cameras also have their problems. If they have a diaphragm, you cannot (and should not) stop down too much. Lenses with short focal lengths are subject to diffraction problems when stopped down too much. Apertures such as f/64 are definitely out of the question. Therefore, compact cameras usually can only be stopped down to f/16. Many small cameras don't have a diaphragm at all, but use a neutral density (ND) filter in case the scene is too bright. So the camera is always working at maximum aperture with the lowest possible depth of field *(DOF)*.

Focus stacking is a technique used to solve DOF problems. Images taken with varying subject distances are combined in such a way that the sharp areas from the single images are visible and the unsharp areas are invisible. The result is a composite image with a very large DOF. This technique is especially popular in macro photography.

The images can be combined manually by masking the unsharp areas of each image and then stacking the images together. However, the results are often unsatisfying because the transitions are not good. Heavy retouching is required. A better option is to use specialized programs, so-called focus stackers, for this task. Programs such as the free command line

programs *enfuse* and *tufuse,* the free programs *CombineZP* and *Picolay,* and also the commercial programs *Helicon Focus, PhotoAcute Studio,* and *Zerene Stacker* can be used. *Photoshop CS4* now offers focus stacking, too.

Taking the series of single photos requires a bit of planning. You must make sure that the different DOF areas of the images overlap nicely so that they can easily be combined into one large DOF area. Fortunately, the CHDK offers a *DOF Calculator* (section 4.2.8). Switch it on by going to **ALT** > **MENU** > *OSD Parameters* > *DOF Calculator* and enabling the entry *Show DOF Calculator*. To be on the safe side, enable the entry *Use EXIF Subj. Dist. (PC65),* too. You will probably get a reading that is a little too short, but that is better than a reading that is a little too long. Then put your camera on a tripod, point the camera at the closest point of your subject, focus, and note down the following readings:

- Subject Distance (`S` or `SD`)
- Depth of Field (`DOF`)

Also, measure the distance to the farthest point of your subject (`F`). `F-S` is your desired depth of field. Now you know the number of required shots. It can be computed from the formula `2*(F-S)/DOF`. For example: given a distance of 60 mm to the nearest point of your subject, a distance of 120 mm to the farthest point of your subject, and a DOF of 12 mm, you would end up with `2*(120-60)/12 = 10` exposures.

If you don't use manual focus to do the measurements, you may want to switch your *AF Frame* to **Center** for precise focusing.

Now you can configure the camera. If your camera features a manual focusing mode, switch it on. Now dial in the following values:

- To enable bracketing, you must set *Disable Overrides* in the submenu *Extra Photo Operations* to **Off** (section 4.3.1) if you had enabled the option *Include AutoISO & Bracketing* (in the same submenu). Do not use *Overrides* for the subject distance.
- Now go to **ALT** > **MENU** > *Extra Photo Operations* > *Bracketing in Continuous Mode.* Dial half of the measured `DOF` into the menu entry *Subj.Dist. Bracket Value*. You need to set the subentry *Value Factor* unequal **Off**. For example, if you measured a `DOF` of 100 mm, you could set the *Value Factor* to 10 mm and the *Subj.Dist.Bracket Value* to 5. This would result in a bracketing value of 50 mm.
- As *Bracketing Type* select "+". The camera will then make exposures with increasing subject distance. For example, if you start at a subject distance of 500 mm and a bracket value of 50 mm, the camera will take exposures at 500 mm, 550 mm, 600 mm, and so on.
- Then switch the camera to *Continuous* mode. Again, focus on the nearest point of your subject. Holding the shutter button half-pressed, turn

the camera a bit to put your subject nicely into the frame. Start shooting by pressing the shutter button down and holding it there. Either count the required number of pictures until you release the shutter, or simply watch the display and observe how the sharpness area moves beyond the farthest point of the subject matter.

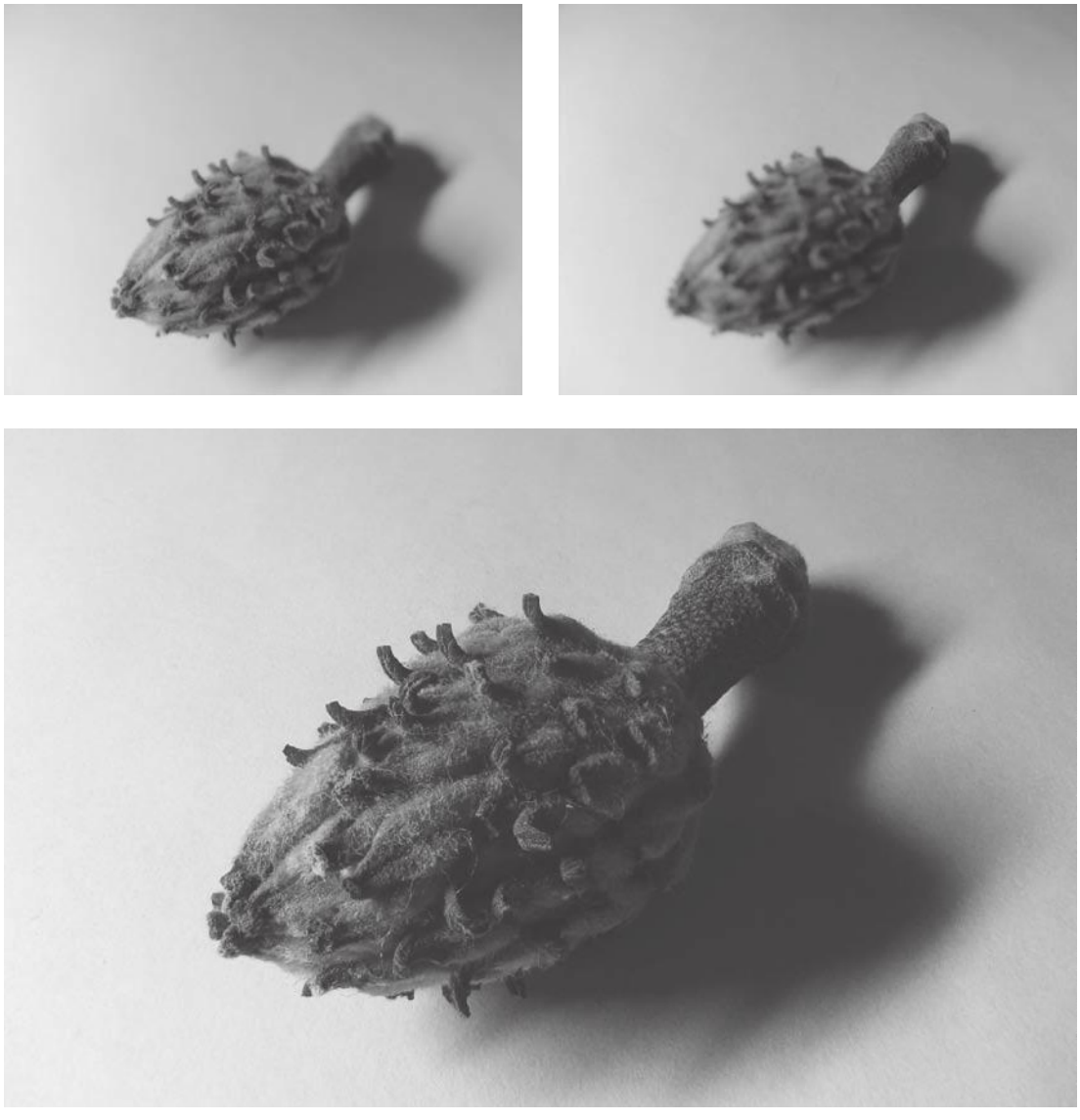


Figure 4-39

Magnolia. Canon Digital Elph 1100SD (Ixus 80 IS). 1/60 sec, f/2.8, ISO 160. f=6.2 mm (~38 mm). Top left is the first shot in a series of nine images; top right is the last. At a near distance of 142 mm, the measured DOF was 15 mm. The *Subj.Dist.Bracket Value* was therefore set to 7 mm. The whole covered range of sharpness is 10 x 7 = 70 mm. Image composition was performed with CombineZP. No artifacts were created during composition, so retouching could remain at a minimum (some dust spots on the surface).

4.7 Edge overlay

The CHDK *Edge Overlay* function is able to extract the edges from an image as soon as you half-press the shutter button and overlay the resulting diagram with the current content of the display. This allows viewing the previous shot and the current shot in context, making it easy to register images and apply techniques such as onion skinning[7].

Figure 4-40
Edge overlay during a panorama shot. When the first image is taken, the CHDK extracts the edges of the subject. Then the camera is turned. Because the first image is still visible as a "ghost", the photographer gets the opportunity to register the next shot with the previous one.

Edge overlay has applications in panorama photography, stereo photography (with a single camera), time-lapse photography, bracketing photography, and more.

Of course, it is possible to choose the color of the edge overlay (menu entry *Edge overlay color*) or to show the edge overlay in *Replay* mode. A threshold value *(Edge overlay threshold)* determines which brightness differences are regarded as edge. The lower this value, the more and wider edge lines you get; higher values create fewer and finer lines.

With the menu entry *Lock Edge Overlay,* you can temporarily freeze an existing edge diagram so that it is not replaced by a new one when you half-press the shutter button. Typically, you would use this option when you want to register several images with one and the same reference image.

Edge overlays can be saved *(Save Edge Overlay)* to the folder `CHDK/EDGE/` on the memory card. This allows you to use them at a later time. The function *Load Edge Overlay* opens the file browser (section 4.10.1), allowing you to select an edge overlay file.

For example, if you want to photograph a tree at different times throughout the year, you could save an edge diagram of the first picture and reload it when you return to the scene. By registering the diagram with the image in the display, you can be sure that the tree will always have the same position in all images.

The zoom level is saved with the edge overlay as well, so when you reload an edge overlay *(Load Edge Overlay),* the camera is optionally set to the stored zoom level if option *Load+Set Zoom* is enabled.

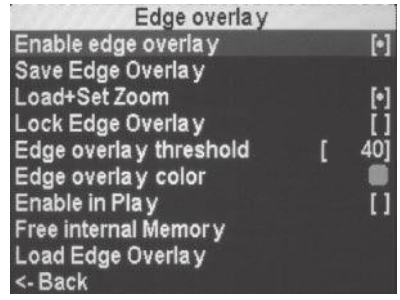


Figure 4-41
The *Edge overlay* submenu

7 The term onion skinning refers to a technique of creating animated cartoons by overlaying individual frames drawn on onionskin paper.

The *Edge Overlay* function allocates some internal memory. When an overlay is no longer needed, it is possible to regain the memory with the function *Free internal memory*. Normally, this is only necessary on cameras where internal memory is scarce.

4.8 More video options

The video functions of small cameras are often overlooked by photographers. We take photographs, not video clips, right? Well, with the CHDK activated you may change your mind.

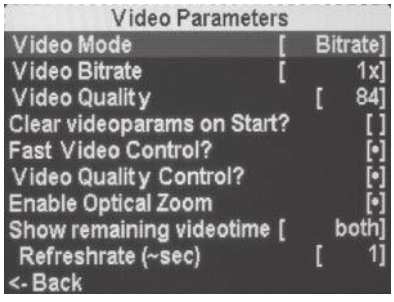


Figure 4-42
The submenu for *Video Parameters*

Video sequences are quite memory-hungry. That is the reason why the camera compresses all video frames. Doing so saves a large amount of memory but also reduces quality. To support different requirements, the camera offers different video modes: *640* (640x480 pixels), *640 LP,* and *320* (320x240 pixels). Compared to *640,* the *640 LP* mode generates only half the amount of data, so that the maximum length of a video sequence is doubled—albeit at a lower quality. The *320* mode produces only a third of the data, resulting in a tripled maximum length. Some cameras also offer a *Compact mode* (160x120 pixels for sending videos by email) that reduces the amount of data by around a factor of nine. Here, the clip length is restricted to three minutes.

Generally, the clip length is restricted to 4 GB or one hour, whichever is shorter. Some older cameras only allow for 1 GB, but the CHDK removes that limit. In addition, the CHDK allows you extended and fine-grained control over video quality and compression. You have the option (in menu entry *Video Mode*) to specify the quality, either in terms of compression *(CBR mode, Constant Bitrate)* or quality *(VBR mode, Variable Bitrate)*. When the quality mode is used, the quality remains constant and the camera automatically adapts the bitrate to the quality setting and subject matter. A higher quality setting leads, of course, to a higher bitrate—and a subject with finer details as well.

- The *Video Bitrate* can be set to the values 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.5 and 3.
- The *Video Quality* can be set to a value between 1 and 99, with 99 representing the best quality with the least compression. The camera's native *640 mode* is equivalent to a setting of 70–75.

So, you can in fact get a better quality than what the native video mode provides. In particular, close-ups and out-of-focus objects will look more natural. However, it may well be that the camera or the memory card is unable to handle such a large amount of data. In this case, the CHDK shows a warning sign (**!**) on the display. When the camera's internal buffers overflow, the video sequence is aborted in a controlled manner.

Let's discuss the other parameters:

- **Clear Video Params on Start?** Enable this entry if you want to reset the camera's video facility to its native state on power-up.
- **Fast Video Control?** This allows you to pause the video recording with the LEFT button. You can then resume with the RIGHT button. **Note:** This does not work with all cameras.
- **Video Quality Control?** This allows you to change the video quality or bitrate DURING recording with the UP (increase) or DOWN (decrease) buttons. Again, this does not work with all cameras.
- **Enable Optical Zoom**. Many of the supported cameras do not allow the use of optical zoom during recording, but rather restrict the zoom to the range of the *Digital Zoom*. *Digital Zoom* is not a bad thing when recording video; it is silent, and the quality does not suffer much because the camera sensor has a much higher resolution than what is needed for video. Still, in combination with the *Optical Zoom,* you can get those long-range zooming shots. For instance, when the optical zoom range is 3 and the digital zoom range is 4, you would get a zoom range of 12. Unfortunately, for most cameras, the manufacturer has disabled the *Optical Zoom* during recording, probably to avoid complaints from customers who are irritated about the noise from the focus mechanism in the soundtrack. The CHDK allows you to lift that restriction and enjoy a wide zoom range. And the noise? Well, some editing of the soundtrack is required. Unfortunately, the *Enable Optical Zoom* function does not work on all cameras (it doesn't on my SD1100, for example).
- **Mute During Zooming**. This entry is not present on all cameras. It allows you to mute the sound recording while zooming and thus avoid recording the noise generated by the focus mechanism.
- **AF key**. This entry is not present on all cameras. It allows assigning a key (**shutter halfpress** or **FUNC/SET**) to the autofocus function. Pressing this button during recording will cause an autofocus scan. For example, when you start your sequence at infinity and pan on a close subject, you can press this button and refocus on that subject.
- **Show Remaining Videotime**. This entry controls the OSD during video recording. You can hide the remaining recording time *(Don't),* show it *(hh:mm:ss),* show the current bitrate in kilobytes per second *(KB/s),* or show both remaining time and bitrate.
- **Refresh Rate (~sec)**. This entry controls how often the remaining video time is updated. Longer intervals result in more precise values; shorter intervals are more up-to-date.

Your camera's native video options probably include an option for creating time-lapse movies. This feature is quite limited; on my SD1100 I can select between a one and a two-second interval. In section 5.7.1 we will discuss

how to create advanced time-lapse movies with the help of the CHDK—in HD quality, of course.

4.9 Remote control

Most Canon compact digital cameras do not come with a remote control—neither an infrared (IR) remote control nor a wire-bound unit. In contrast, some other manufacturers do provide such a unit—in most cases, an IR remote control. Such controls, however, have a few disadvantages: The distance between the IR sender and the camera is limited, and the IR sender must usually be positioned in front of the camera.

Many more advanced cameras, such as DSLRs provide the option of controlling the camera through the USB port. The advantage here is that different devices, such as wire-bound remote controls or radio frequency (RF) based controls, can be connected to the camera.

4.9.1 CHDK remote control functions

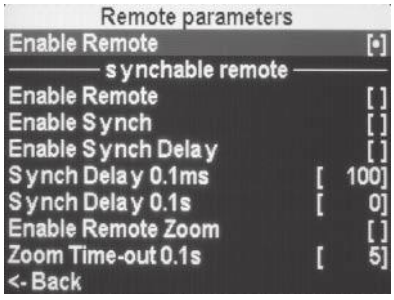


Figure 4-43
The *Remote parameters* submenu can be reached via ALT > *Miscellaneous Stuff* > *Remote parameters*. It has two sections: one for simple USB switches, as shown in section 4.9.2, and one for remote controls supporting the Ricoh CA1 protocol.

The CHDK equips most Canon compact cameras with exactly such capabilities. You have the choice between simple devices (battery and switch) and more complex devices that support multiple functions or that can hook up to a RC control system used for model airplanes and the like.

The native mode for simple USB switches is enabled in the upper section of the *Remote parameters* submenu:

- *Enable Remote*. Enables the USB remote function. When enabled, the camera can be controlled through the USB port. This setting applies both to scriptless and script mode.
  Note: You must disable this entry if you want to transfer images through the USB port to a PC. Otherwise, the PC would be recognized by the camera as a remote control.

In scriptless mode, the camera must be in normal shooting mode and be operated through the button of the remote control. You can make the camera focus with a short click to the button of the remote control (equivalent to a shutter button half-press). To take a shot, you must perform a "reverse click"—releasing the button and immediately pressing it down again.

On some cameras, the USB remote can even be used in playback mode: clicking the remote control will advance through the pictures (in reverse order) just as if you clicked the LEFT button. This can be nice; you can stand back while your audience gathers around the display.

In scripted mode, the camera is in the *<Alt>* state and is running a script. A signal at the USB port is recorded as a button click under the button name "remote", or can be intercepted with a special command. We will discuss these scripts in section 5.7.5.

4.9.2 Building a simple remote control

In the USB specification, each pin has a special purpose. While electrical ground is assigned to *Pin 4, Pin 1* is used for the supply voltage. The CHDK interprets the presence or absence of the supply voltage as a signal. You can test this easily even without a proper remote control. First, enable the *Remote* function, then connect a USB cable to the camera and quickly connect-disconnect-connect the other end to the USB port of a PC. Because the PC delivers supply voltage at *Pin 1,* the CHDK will interpret this as a signal from a remote control and will fire:

USB Pins		
Pin	**Purpose**	**Wire Color**
1	VCC (+5V)	Red
2	Data –	White
3	Data +	Green
4	Ground	Black

A USB cable release is relatively easy to build. A small battery is needed to supply a voltage of not more than 5V (**Warning:** the USB specification allows for a maximum of 5V. Your camera might be damaged if you use a higher voltage.) The minimum voltage required depends on the camera. You may find it on the *CameraFeatures* reference page of the CHDK wiki (http://chdk.wikia.com/wiki/CameraFeatures). Some cameras are happy with 3 volts (or even less), but other cameras require up to 4.5 volts.

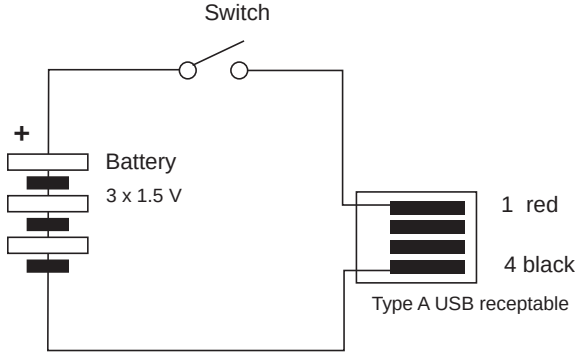


Figure 4-44
Wiring diagram for a USB remote switch. The battery here delivers 4.5 volts, which should be sufficient for any camera. This voltage is safe even if the camera operates at a lower voltage. The USB specifications require the camera to tolerate a voltage of 5V or less at Pin 1. But be sure to get the polarity of the battery right: wrong polarity could damage your camera!

So, what do you need to build a cable release? First, a USB cable that you can sacrifice, then a switch and a battery. A good option is a small LED flashlight that you can buy for a few cents. With such a flashlight, a *USB Type-A to mini-A* cable, and some soldering skills, you can build your own remote control. For an example, see https://sites.google.com/site/canonremotehowto/. For building a DIY infrared remote control, see www.instructables.com/id/Remote-for-Canon-Compact-Cameras/.

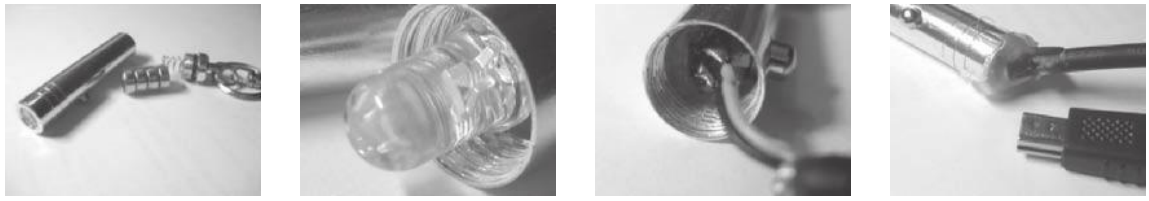


Figure 4-45
A cheap LED flashlight (1) is the basis for a homemade USB remote control. It's important that it have a switch and not be turned on or off by twisting its head. This one has four batteries resulting in a voltage of 6V. One of the batteries must be removed because we only need 4.5V. The spring at the bottom end must be stretched a bit to bridge the gap of the missing battery. The LED (2) is twisted off with tweezers. A voltmeter is used to test which contacts are positive and negative. Then the larger USB plug is removed from a spare mini USB cable. The head of the flashlight is stripped over the cable; then the red wire is soldered to the positive contact (3) and the black wire to the negative contact. The flashlight head is screwed on again and sealed with epoxy or hot glue (4).

4.9.3 SDM functions

The remaining functions in the *Remote parameters* submenu under the header *"synchable remote"* (Figure 4-43) were originally developed in the *Stereo Data Maker (SDM)* project (section 7.5) but are now also part of the CHDK. These functions require a *Ricoh CA1* or a compatible remote control.

- *Enable Remote*. This entry enables the SDM functionality in **scriptless** mode. The camera can now be controlled through the *Ricoh CA-1*.
- *Enable Synch*. Allows synchronizing two or more cameras through linked remote controls. This option is typically used for stereo or matrix photography (section 7.7.4).
- *Enable Synch Delay*. Enables delay settings for synchronized cameras down to 1/10,000 sec. Although not recommended, it is possible to synchronize different camera models by using a *Synch Delay* with one camera (section 7.7.4).
- *Synch Delay 0.1ms*. A delay can be dialed in down to 1/10,000 sec. This is added to the value in the following entry.
- *Synch Delay 0.1s*. A delay can be dialed in down to 1/10 sec. This is added to the value in the previous entry.

- *Enable Remote Zoom*. Allows controlling the zoom of multiple synchronized cameras through the remote control on cameras with a single zoom switch or rocker. To do so, the camera must first be brought into remote zoom mode. This is done by manually clicking the zoom switch quickly. The blue <Alt> button should light up, and the zoom can now be operated using the button on the remote. When it reaches the longest or shortest focal length, the zoom direction will reverse. You can exit the zoom mode by half-pressing the shutter button, by half-pressing the Ricoh switch, or by simply waiting for time-out (see below). Afterwards, the remote control can be used to fire the shutter.
- *Zoom Time-out 0.1s*. Specifies a zoom time-out value in seconds. (The label specifies an incorrect time unit of 0.1 sec. The actual time unit is 1 sec.) After this time, the camera leaves the zoom mode and returns to normal shooting mode. Time-out values between 2 and 10 seconds are possible.

4.9.4 Extra hardware

It isn't really necessary to learn soldering to get your own remote control device. Some vendors are offering devices that can hook up to the camera's USB port and that are supported by the CHDK.

- The *Ricoh CA1* USB remote control works with the CHDK functions listed under "synchable remote". The release button has a half-pressed and a pressed position. A double click is therefore not necessary to shoot a picture. A single 1.5V AAA cell powers the device. A built-in DC-DC converter pushes the 1.5V up to the required signal voltage.
- The *GentLED* devices from *Gentles Limited* (www.gentles.ltd.uk/gentled/) allow remote control of cameras by wire, infrared, or RC control systems. These products support both the Ricoh CA1 and the native CHDK protocol. Using the native protocol, these devices can encode different functions using different pulse lengths that can be evaluated by a script. For example, the different joysticks of an RC remote control can be assigned to different functions, such as shooting, zooming, or focusing.

4.9.5 Tethered shooting?

Tethered shooting is the ability to transfer images immediately to a connected PC without saving them on a memory card. In a studio situation or for time-lapse work, this makes a lot of sense because you aren't limited by the capacity of the card. You also save the extra step of transferring images.

Does the CHDK support tethered shooting? Currently, no. If you are looking for this functionality, you may want to check out the program *PSRemote* from *Breeze systems*. If your camera is on the manufacturer's list of supported *Powershot* models, you will be able to control the camera from the PC and upload images immediately. You even get a remote live viewfinder on the PC screen, along with some of the tricks the CHDK is famous for, such as grids, automatic bracketing, and overlays.

If you own an older Powershot model, you may want to try Canon's own remote control software, *Remote Capture,* that comes for free. Unfortunately, Canon has discontinued this software; the latest version is from 2004.

Another option is to use a wireless SD card such as the *Eye-Fi card* (www.eye.fi) for automatic transfer from camera to computer.

Finally, with the CHDK spin-off *Stereo Data Maker* (SDM), you can do semi-tethered shooting. Here, a script can alternate between shooting and uploading images to the PC-based WIA loader (section 7.6).

4.10 Utilities

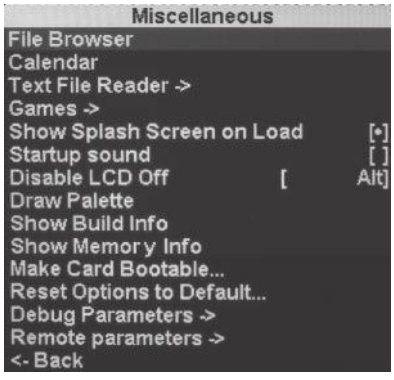


Figure 4-46
The *Miscellaneous* submenu allows accessing utilities and also provides various display options.

Apart from photographic functions, the CHDK also implements some utilities and fun programs such as a file browser, file reader, calendar, and games. These programs can be accessed via ALT > MENU > *Miscellaneous Stuff*.

4.10.1 File browser

The file browser is invoked via ALT > MENU > *Miscellaneous Stuff* > *File Browser*. It is used to inspect the content of the memory card and to manage the files and folders found there (Figure 4-47).

Navigation in the file browser is easy. Simply use the UP and DOWN buttons to select a folder, and press FUNC/SET to move into that folder. The zoom rocker can be used to scroll a whole page up or down. Navigating backwards into a parent folder is performed by selecting the entry `../` and pressing FUNC/SET.

Selected files and folders can be deleted with the DISP button. After pressing this button, you are asked if you want to delete the selected file. Use the LEFT button to highlight YES and press FUNC/SET to delete the file. Folders can only be deleted if they don't contain nonempty subfolders. If they do, first navigate inside the folder and delete the subfolders. Then return to the parent folder to delete the folder in question.

The RIGHT button can be used to mark files. Another click on the RIGHT button removes the mark from marked files. Marked files play a role in the

functions from the context menu. This is invoked by pressing the LEFT button. Another click on the LEFT button closes the context menu.

Figure 4-47
The functions of the context menu apply to marked files only. Here, two files are marked (appearing here as grayed).

Most of the functions of the context menu are related to RAW file processing. We have already discussed that function in section 4.5.6. Only a few general file management functions remain:

- *Cut*. Cuts all marked files from the current folder and stores them in the clipboard. If no file is marked, the selected file is cut.
- *Copy*. Copies all marked files from the current folder and stores them in the clipboard. If no file is marked, the selected file is copied.
- *Paste*. This function is only available if the clipboard does contain files. It will copy the files in the clipboard into the current folder.
- *Delete*. Deletes all marked files. If no file is marked, the selected file is deleted.
- *Select Inverse*. Unmarked files are marked, and marked files are unmarked.

4.10.2 Text file reader

The *Text File Reader* is typically used to look into text files stored on the memory card. For example, you might want to look into the comments of a script file, or read a README file or some other documentation. As its name says, the *Text File Reader* only allows reading files, not modifying them. It is also restricted to plain text files—PDFs, HTML pages, or *Office* files are not in the scope of the *Text File Reader*.

When starting the *Text File Reader* with the function *Open New File...* you will first see a file browser showing the contents of the folder CHDK/BOOKS/. Now navigate to the file that you want to open and press FUNC/SET. If you want to reopen the file later, you can invoke *Open Last Opened File*.

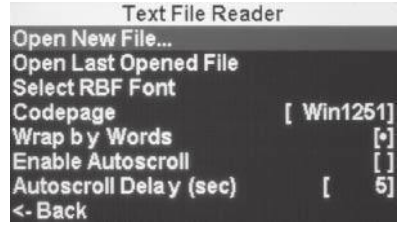


Figure 4-48
The *Text File Reader* allows opening and reading text files. It has its own settings for fonts and codepage, and supports word wrapping and auto scroll.

When the file is showing, you can scroll downwards with the **RIGHT** button, the **DOWN** button, and the **ZOOM_IN** button. You can scroll upwards with the **LEFT** button, the **UP** button, and the **ZOOM_OUT** button. The **MENU** button leads back to the *Text File Reader* submenu:

- Here, the function *Select RBF Font* opens the font browser (section 4.2.2). A larger font may be easier to read, but it also reduces the amount of text that can be displayed on one page. Not all fonts support all national characters. So, if national characters (umlauts and the like) don't display correctly, you may want to try another font.
- The *Codepage* option allows selecting between *Win1251* and *DOS* codepages. If a text file does not display correctly and shows strange characters, you might want to try the other codepage.
- The reader always wraps lines that are too long to display into the next line. When the option *Wrap by Words* is set, it will try not to wrap within words but to honor word boundaries.
- Finally, the *Enable Autoscroll* option can make the reader scroll automatically. Every time the specified *Delay* expires, the reader scrolls down one page.

4.10.3 Getting information about the camera

Under **ALT** > **MENU** > *Miscellaneous Stuff* there are a few useful functions that display information about the camera:

- *Show build info*. Shows the current version of the CHDK in use and related information.
- *Show memory info*. Displays free memory available in the camera RAM and the space used by the CHDK.

More information is found in the *Debug* submenu. Most of this information supports CHDK developers, but some of it may be useful for "normal" users. **Warning:** Before playing around with the options of this submenu, make a backup of your memory card—and in particular, a backup of the CHDK configuration file `CHDK/CCHDK.CFG`. Entering bad values can cause the CHDK to fail at start-up. In this case, you must be able to restore the configuration file from an earlier version (section A.2).

- *Debug Data display*. This can be used for displaying *Property Cases* (section 5.6), parameters, and operating system tasks (only when running under *VxWorks*) on the OSD. Because there are many of these values, the display can be positioned to a section within the values by setting the entries *PropCasePage* and *Task List start* to appropriate values.

- *Show Misc Values*. Shows some values for service and programming purposes.
- *Memory Browser*. Invokes the memory browser that displays memory content on the screen. The memory address can be changed with the LEFT and RIGHT buttons; the DISP button changes the step width *(Incr)*. The MENU button returns to the *Debug* submenu.
- *Benchmark*. Launches a benchmark utility for evaluating the performance of the camera memory, display, and memory card. Simply press FUNC/SET to start the benchmark, sit back, and wait. Press MENU to return to the *Debug* submenu.
- *ALT +/- debug action*. Assigns specific debug functions to the +/- key (DISP key on SD/Ixus). If set to **DmpRAM**, the key will write a complete memory dump to the card. If set to **Page**, the key will scroll through *Property Cases* and *Parameters* (see above) and toggle the scrolling direction when pressed twice. If set to **CmpProp**, the key will compare *Property Cases*. To use this function, first press the +/- key in *<Alt>* mode; then leave the *<Alt>* mode and change some camera settings; then return to the *<Alt>* mode and press the +/- key again. The display will now show up to 12 *Property Case* values that have changed.

4.11 Novelty

Finally, there are a few functions that seem to have absolutely nothing to do with photography. But maybe you want to play a few games with your camera while you wait for the right shooting light.

4.11.1 Games

Games on a camera? Why not! If you can take photos with a *Nintendo Gameboy,* why not play games with your camera? So, the CHDK offers four games: *Connect4, Mastermind, Reversi,* and *Sokoban*. You'll find all of these games in the submenu ALT > MENU > *Miscellaneous Stuff* > *Games*.

Before invoking a game, you must put the camera into *Replay* mode. The game starts right away, and you can use the following keys to interact:

- *Reversi:* UP, DOWN, LEFT, RIGHT: Move cursor; FUNC/SET: Draw/Restart Game; DISPLAY: Camera against camera; MENU: back to the *Games* submenu.
- *Sokoban:* UP, DOWN, LEFT, RIGHT: Move; FUNC/SET: Change level (only at start of a game); DISPLAY: Restart current level; ZOOM_OUT: Undo moves; ZOOM_IN: Redo moves; MENU: back to the *Games* submenu.

- *Connect4:* **LEFT**, **RIGHT**: Select column; **FUNC/SET**: Drop ball/Restart game; **MENU**: back to the *Games* submenu.
- *Mastermind:* **LEFT**, **RIGHT**: Select column; **UP**, **DOWN**: Select color; **FUNC/SET**: Next row/Restart game; **MENU**: back to the *Games* submenu.

While trying to beat your camera, you should now and then have a look at the battery level displayed in the lower part of the info section. Maybe you'll still need some battery power when the shooting light turns perfect.

4.11.2 Flashlight

The *Flashlight* option (**ALT** > **MENU** > *Miscellaneous Stuff* > *Flashlight*) is only available on cameras with a fold-out display. When in *Record* mode, the display turns out, swivels towards the scene, and turns all-white. This allows you to illuminate the scene during set-up, or simply use it as a flashlight to find your way in the dark. However, it cannot act as a light source when you actually take a shot; the camera turns the display dark at that very moment.

4.12 The CHDK configuration file

All of the choices that you make in the CHDK menus are stored on your memory card in the file `CHDK/CCHDK.CFG`. This file can be edited on the PC, too. The *CHDK Config File Editor (CFGEDIT)* is a utility that runs on Windows, Linux, and Mac OSX. It allows editing this file conveniently on the large screen of your desktop or laptop computer. All you need is *Java Runtime* installed on your computer. Then simply download the editor and double-click on the JAR file to execute it.

Because all configuration options are stored in a single file, it is relatively easy to switch between different configurations. The script `configsw.lua` developed in section 5.7.6 allows doing just that. However, this script will only switch the CHDK configuration, not the native camera configuration.

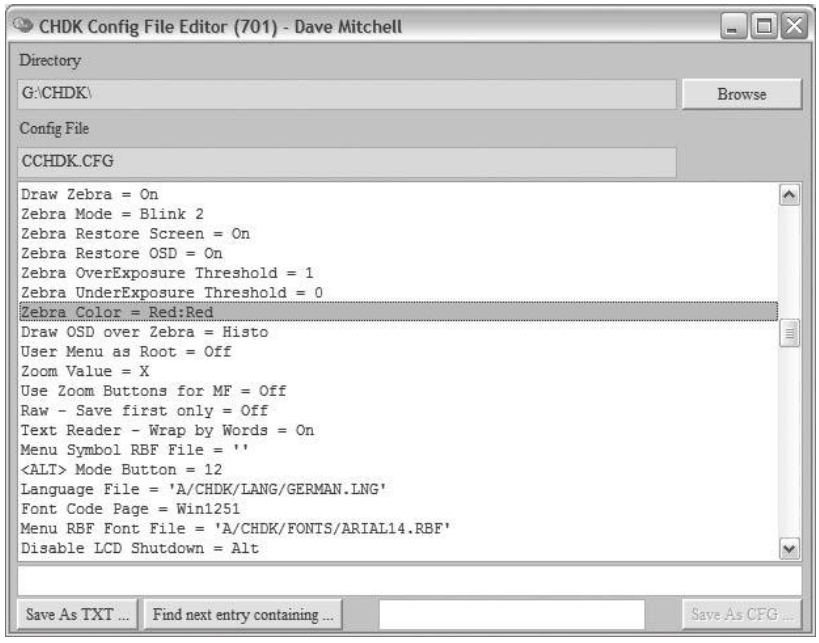


Figure 4-49
The *CHDK Config File Editor* in action. A double-click on an entry opens a specific editor for the selected entry value.

5 Scripting

The ability to control the camera with a script is one of the outstanding features of the CHDK. With scripting, you can automate almost every photographic task. You can shoot time series, react to motion, shoot all sorts of bracketing series for high dynamic range photography (HDR) or increased depth of field, and much more. In fact, many users say that camera automation via scripting has been the main reason they installed the CHDK. Therefore, we will discuss scripting in detail and also present some sample scripts[1] from different areas where scripting can be useful.

On the Internet, and in particular on the CHDK website, you will find a large number of premanufactured scripts—mostly for time series, motion detection, and bracketing. Many of these scripts have only been tested with a few camera types. So, before using such a script, try it out and establish whether it works properly with your camera. In particular, older scripts written for the DIGIC II cameras need some adaptations to run on DIGIC III or DIGIC IV cameras, and vice versa. Therefore, even if you do not plan to write your own scripts, it is worth taking a closer look at CHDK script development.

Newer builds of the CHDK support two scripting languages: the simple *uBasic* and the more advanced *Lua* language. The majority of existing scripts are written in *uBasic* simply because it's been supported by the CHDK since the very first version. Because *uBasic* is simple, it is well suited for beginners. *Lua,* on the other hand, requires some knowledge about basic concepts of computer programming.

5.1 Launching and configuring scripts

Scripts are stored on the memory card in the folder `CHDK/SCRIPTS/` or a subfolder. To load a script, first switch to the *<ALT>* mode, then press the **FUNC/SET** button[2]. Now the script menu will display. Select *Load Script from File ...* and press **FUNC/SET** again. The file browser (section 4.10.1) will be shown positioned at folder `CHDK/SCRIPTS/`. Navigate to the script that you want to load and press **FUNC/SET** again. The script is now loaded and can

1 All scripts are found on the book CD.
2 If you had set the option *User Menu Enable* to *OnDirect,* you first need to close the user menu by pressing the **MENU** button.

be executed immediately by pressing the shutter button. Pressing the shutter button another time will interrupt the script execution immediately.

The next time you turn on the camera and the CHDK is activated, the selected script will be loaded automatically. To execute it, it is sufficient to switch to the *<ALT>* mode and press the shutter button. It is even possible to run a script automatically at start-up. This is enabled by setting the entry *Autostart* in the script menu to *On*. However, you should do this only with well-tested scripts—otherwise, you could run into an error every time you turn on the camera.

The *Autostart* feature can be used to initialize the camera with custom settings when it is switched on. For example, in automatic mode, my little SD1100 IS always switches the flash to *AUTO*. (In manual mode, it remembers the flash mode of the previous session.) The camera offers no option to always start with the flash switched off. This is unfortunate—I like available light shots much more than shots with the built-in flash. Sometimes, in low-light conditions, I forget to switch the flash off—the flash fires, and the mood is gone.

Here, the CHDK can help. A tiny script performs the necessary key presses to switch the flash off. By loading the script `flashoff.lua` and setting the *Autostart* parameter to *On,* I can make sure that this script is executed at start-up and that I always start with a disabled flash. Here is the script, written in *Lua*:

```
--[[
@title Flash off
]]
while get_flash_mode() ~= 2 do
     click("right")
     click("right")
     click("set")
   sleep(10)
end
exit_alt()
```

It's dead simple (apart from the header, which we will discuss later). While the function `get_flash_mode()` does not return the value 2 (this value indicates that the flash is switched off, see section 5.5.5), the script continues to bring up the camera's flash menu with a click on the **RIGHT** button. Then it clicks **RIGHT** again to change the flash mode and confirms the change with a click on **FUNC/SET**. At the end of this sequence is a `sleep(10)` instruction, sending the script to sleep for 10 milliseconds and allowing the native camera tasks to do their work.

The final instruction is necessary because otherwise the script would leave the camera in *<ALT>* mode where no shooting is possible. Of course,

as soon as you load another script, the new script will run at start-up, so it is necessary to reload `flashoff.lua` before switching off.

Many scripts can be configured with parameters. These parameters are defined in the header section of the script. After the script has been loaded, they are listed in the bottom section of the CHDK *Script* menu. The parameter values shown there can be modified with the **LEFT**, **RIGHT**, and **FUNC/SET** keys. If a script has many parameters, it can be tedious to set up all the parameters before running the script. The CHDK therefore offers the possibility to store frequently used parameter combinations in parameter sets. Ten of those sets exist (0–9). By modifying the value of the menu entry *Parameters set,* you can easily switch among different parameter combinations.

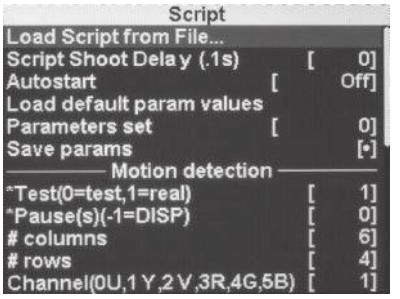


Figure 5-1
The CHDK *Script* menu. The upper half is used for general script configuration, such as script loading, delaying the execution of the script for a specified number of 1/10 secs, resetting to the default parameter values, and switching and saving parameter sets. The lower half is reserved for the script parameters. Here are some of the parameters from the motion detection script discussed in section 5.7.3

5.2 uBasic

The original *uBasic* interpreter was written by the Swedish programmer *Adam Dunkels*. He said, "I've always wanted to write a really small BASIC interpreter. So I sat down for an hour or two and did it." It's probably true. *uBasic* is tiny, and there are only a few commands to learn. The original *uBasic* interpreter didn't even understand labels—instead you specified line numbers as `GOTO` targets, just like in the days of the legendary C64. Labels were later added by *Pablo d'Angelo* who is also the main contributor for the *Hugin panorama stitcher*.

So, let's jump right into another small script and see how *uBasic* looks these days. (A more systematic introduction into *uBasic* is given in the following sections.) The following script implements an electronic *magnifying glass* by switching the camera into the *Digital Macro* mode and setting a predefined magnification level. Also, most of the info texts are hidden from the display.

```
@title Digital Magnifier
@param m Initial Magnification
@default m 3
```

This is the header section of the script. The title will appear at the bottom of the display after the script is loaded. You should avoid using more than 24 characters; otherwise, the title would override the *<ALT>* indicator.

The title declaration is followed by the declaration of the parameters providing input to the script. With the `@param` instruction, you specify a variable name and a parameter description. Variable names for parameters always consist of a single lower case letter. In newer CHDK versions, you can have variable names from `a`–`z`, in older versions only `a`–`j`. The parameter description is shown in the script menu (**ALT** > **FUNC/SET**) below the

script title. End users are free to supply a value or to accept the default value. The `@default` instruction specifies this default value.

Because the script needs some time to set the camera up, we use that time by giving the user something to read—for example, the usage of the camera buttons during loupe mode:

```
print "Press MENU to cancel"
print "Zoom = magnification"
print "Shutter/2 = refocus"
```

This is good practice. When you allow for some user interaction, always explain how to use the controls. Sometimes scripts are not used for a long time, so the user might have forgotten how to operate the script.

```
rem init
if m < 1 then m = 3
```

Comment lines begin with the token `rem`. Comment lines are not executed and do not affect the state of the camera. Their purpose is documentation. Nevertheless, many comments can slow down a script. After each script line (including comments), the CHDK pauses for 10 milliseconds to let the camera do its work and look after user inputs. Newer CHDK builds, however, allow for larger comment blocks—they pause only once for up to 100 consecutive comment lines.

During initialization, you should check the validity of the parameters and correct them as necessary. We set the input parameter to its default value if the user entered an invalid value. A magnification of `0` is obviously nonsense.

```
let s = 0
if m > 1 then s = 3
if m > 2 then s = 5
if m > 3 then s = 6
```

From the specified magnification, we compute the zoom step value. This value will be used later for setting the zoom level. This section depends on the camera model; different camera models have different zoom granularity. While A-series cameras have 9 or 15 zoom steps, S-series cameras feature 129 steps. It would be possible to retrieve the number of zoom steps from the camera and come up with a more general logic for selecting the right zoom step, but for now we leave it at that for simplicity.

```
rem DigitalMacro
o = get_display_mode
d = 1
gosub "display"
p = 0
e = -32248
gosub "mode"
```

Then we start to set up the camera. First, we save the current display state into variable `o`, so that when leaving the script we can reset the previous state. We assign the desired display state (**No Info** = 1) to variable `d`. The subroutine `display` will click the **DISP** button until that state is reached. Because *uBasic* subroutines do not support variables and don't return values, we must transfer data between subroutine and caller via variables.

To switch the camera mode, we will work a little bit differently. The variable `p` will count the number of **RIGHT** clicks to reach the *DigitalMacro* mode. We initialize it with 0. Then we assign –32248 to variable `e`. This value represents the *DigitalMacro* mode (section 5.6). The subroutine `mode` will pick up that value; press **FUNC/SET**, **RIGHT**, **FUNC/SET** repeatedly until this mode is reached. Both subroutines (`display` and `mode`) are discussed below. It's important to know that by simulating key presses, we can reach any camera function via a script that we can reach manually. It's good practice not to make assumptions about the initial state of the camera, and to restore the initial state when the script ends.

```
rem set zoom level
q = get_zoom
gosub "zoom"
gosub "focus"
```

Next, the initial zoom level is saved into variable `q`, and the desired zoom level (stored in variable `s`) is set in subroutine zoom. Afterwards, subroutine `focus` is called to focus the camera to the subject matter.

```
cls
rem Event loop until MENU pressed
do
     wait_click
if is_pressed "zoom_in"
          then gosub "zomin"
     if is_pressed "zoom_out"
          then gosub "zomout"
     if is_pressed "shoot_half"
          then gosub "focus"
until is_pressed "menu"
```

After the *DigitalMacro* mode is set up, the magnification is set, and the camera is focused, the screen is cleared with `cls` (so that printout is removed from the display). The digital magnifying glass is now ready for use. Because the script controls all camera functions in the *<ALT>* mode, the user has by default no ability to focus or to change the zoom level. If you want to provide the user with such possibilities, you must implement the necessary interaction.

The script does this within a `do`-loop where it waits for a key click. It analyses this click and—depending on the key pressed—calls the corresponding subroutine. The keys **zoom_in** and **zoom_out** are the zoom rockers, and the key **shoot_half** is the half-pressed shutter button. The loop ends when the **MENU** button is pressed.

```
rem restore camera state
s = q
gosub "zoom"
gosub "restor"
d = o
gosub "display"
rem leave alt mode
exit_alt
end
```

After the **MENU** button is pressed, we clean up behind us and reset the camera to its original state. We do this by assigning the saved values for the zoom level and display state to the transfer variables `s` and `d`, and then calling the subroutines `zoom` and `display` again. For restoring shooting mode, the subroutine `restor` is called. Finally, the script turns the *<ALT>* mode off.

```
:zomin
click "zoom_in"
sleep 200
return

:zomout
click "zoom_out"
sleep 200
return

:focus
press "shoot_half"
do
 r = get_shooting
```

```
until r = 1
release "shoot_half"
return
```

These are the subroutines for user interaction. They simulate button presses and wait (`sleep`) for some milliseconds to give the camera time to perform the task. Zooming in and out is quite simple this way. Focusing is a bit more demanding. First, we have to half-press the shutter button. Instead of using the `click` command that simulates only a short click, we use a `press` command to hold the shutter button half-down. Then we wait until the command `get_shooting` returns the value 1. This indicates that focusing has finished, so we can now release the shutter button.

```
:zoom
r = get_zoom
if r < s then
  for n = r to s
    click "zoom_in"
      sleep 200
  next n
else if r > s then
  for n = s to r
    click "zoom_out"
      sleep 200
  next n
endif
return
```

The subroutine `zoom` is used to set the initial zoom level or to restore it at the end of the script. First, it asks `get_zoom` for the current zoom level. If the desired zoom level in variable `s` is larger than the current one, it clicks the `zoom_in` button repeatedly to adjust for the difference—and vice versa, if the current zoom level is larger than `s`, the button `zoom_out` is clicked instead. This way, we reach the desired zoom level independently from the initial zoom level.

```
:display
r = get_display_mode
while r <> d
    click "display"
    sleep 1250
    r = get_display_mode
wend
return
```

The subroutine `display` asks the camera for the current display state. The necessary clicks are performed only if it differs from the desired state. It then sleeps for a while to let the camera adjust to the new settings. This is done repeatedly until the desired result is reached.

```
:mode
r = get_prop 49
while r <> e
    click "set"
    sleep 1250
    click "right"
    sleep 1250
    click "set"
    sleep 1250
    p = p + 1
    r = get_prop 49
wend
return
```

The subroutine `mode` works in a similar way. The main difference is that it has to perform a whole series of clicks to switch to the next mode (**FUNC/SET**, **RIGHT**, **FUNC/SET**). It also counts the number of mode switches in variable `p`. There are many `sleep` instructions in this subroutine. Switching modes is quite hard work for the camera; in some cases lenses have to be shifted, and so on. Depending on the camera, some tweaking may be necessary for the necessary sleep intervals.

Because there is no direct *uBasic* command for obtaining the current camera mode, we need to retrieve the mode in a different way. Here we read out the value of *Property Case 49* (section 5.6) that reflects the current camera mode. It should be mentioned that this part of the script runs only under the *DryOS* operating system *(Digic III* and *Digic IV)* because property IDs and mode values differ between *VxWorks* and *DryOS*. We will see later that the script language *Lua* provides the necessary capabilities to write platform-independent scripts.

```
:restor
click "set"
sleep 1250
for r = 1 to p
    click "left"
next r
sleep 1250
click "set"
sleep 1250
return
```

Finally, the subroutine `restor` is used to restore the initial mode. It simply does the opposite of subroutine `mode`: instead of stepping right in the mode menu, it steps left. The number of steps is stored in variable `p`.

5.3 uBasic primer

After this initial hands-on contact with *uBasic,* we are now going to explore the language systematically.

5.3.1 Variables

Variables are denoted by a single letter. While early CHDK versions only supported the letters 'a' – 'j', newer versions support the letters 'a' – 'z' and also the upper case letters 'A' – 'Z'. This results in 52 different variables. However, in the script header (parameters), only lower-case variable names are allowed.

Variables can contain both integer and floating-point values. String variables are not supported in *uBasic*. (Some CHDK spin-offs allow for string variables.)

5.3.2 Assignments

Assigning a value to a variable starts with the command **let**, for example:

```
let a = b * c
```

assigns the product of `b` and `c` to `a`. But `let` is just a noise word; we can omit it happily without any problems:

```
a = b * c
```

means the same and is valid *uBasic,* too. In a CHDK script I would prefer this notation because it is shorter.

The following arithmetic operators are supported:

+ Addition
- Subtraction
***** Multiplication
/ Division
% Remainder

5.3.3 Output

The **print** command can be used to show information on the camera display. A `print` command can have an unlimited number of parameters. Parameters are separated by commas or by semicolons. During output, the commas appear as space; the semicolons appear as no space. For example:

```
print "a:", a;"s"
```

will produce the output

```
a: 500s
```

if the variable `a` has the value of 500. Please note that an output line must not exceed 25 characters in length.

The command **print_screen** allows capturing the output into a file.

```
print_screen n
```

with `n > 0` switches capturing on. All output written with `print` will go into file `LOG_n.TXT` in subfolder `CHDK/LOGS/`, too.

`n = 0` switches capturing off. For example,

```
print_screen 5
print "hello"
print_screen 0
print "bye"
```

will produce a file `LOG_0005.TXT` containing the string "hello".

Finally, the command **cls** *(clear screen)* removes the output produced with the `print` command from the display.

5.3.4 Conditional clauses

Conditional clauses are built with the **if ... then ... else ... endif** construct. For example:

```
if a < 0 then
    b = 3
else
   b = 4
endif
```

Relational operators allowed in *uBasic* are:
= (equals)
> (greater than)
< (less than)
<> (not equal)
<= (less than or equal)
>= (greater than or equal)

It is possible, too, to connect several conditions using the operators **and**, **or**, **xor**, and **not**. Parentheses may be used to indicate which operators are executed first:

```
if (a = 1 or a = 2) and b < 0 then ...
```

If there is nothing to do in the `else` case, the `else` clause can be omitted:

```
b = 4
if a < 0 then
    b = 3
endif
```

does just the same and runs faster because each program line in *uBasic* involves a delay of 10 msec.

In simple cases, it is possible to combine all of these statements into one line and omit the `endif`:

```
if a < 0 then b = 3 else b = 4
```

or

```
b = 4
if a < 0 then b = 3
```

Here, the second version is slower because it needs two lines of code.

Note: This shorthand notation without an `endif` can, however, only be used standalone and not nested inside another `if ... then ... else ... endif` construct. In this latter case, you must use the full statement properly closed with `endif`.

5.3.5 Case structures

The **select** statement is a convenient way to avoid complex `if ... then ... else` constructs. It allows executing different statements depending on the value of a variable:

```
select x
case 3,4,7; print "3, 4, or 7"
case 10 to 20; print "10 to 20"
case_else print "neither"
end_select
```

The first `case` statement specifies a list of values that can be of arbitrary length. If the value of `x` matches a list element, the instruction behind the semicolon is executed. In contrast, the second `case` statement specifies a value range. If the value of `x` is within that range, the statement behind the semicolon is executed. Optionally, a `case_else` statement can be used to execute a statement if none of the specified `case` statements did apply. If you want to execute more than one instruction after a `case` statement, you must enclose those instructions into a subroutine and call it with `gosub` (section 5.3.8).

Note: If you also plan to write scripts that run on the *Stereo Data Maker* (SDM, section 7.9), you should avoid the `select` statement because it is not supported in the SDM.

5.3.6 Loops

The **for**-loop is used to increment a counter from a start value to an end value and to execute the instructions within the loop for each counter value:

```
for i = 1 to n
    sleep 10000
    print "Shot", i, "of", n
    shoot
next i
```

takes `n` shots every ten seconds and prints a protocol for each shot on the display. As you can see, the `for`-construct is closed with the command **next** specifying the variable to be incremented.

By default, the increment value of the `for`-construct is 1. By using the **step** clause, however, it is possible to specify a different increment value:

```
for i = 1 to 10 step 3
    print i
next i
```

will print the values 1, 4, 7, 10.

Another loop construct is the **while ... wend** construct. Here, you have to organize your counters by yourself. while ... wend loops run until an arbitrary logical condition fails.

```
i = 1
while i <= n
    sleep 10000
    print "Shot", i, "of", n
    shoot
    i = i + 1
wend
```

does exactly the same as the for-loop shown before.

The construct **do ... until** is quite similar:

```
i = 1
do
    sleep 10000
    print "Shot", i, "of", n
    shoot
    i = i + 1
until i > n
```

The difference with the while ... wend construct is that the loop body is executed at least once because the condition is checked at the end of the loop. The while ... wend construct may not execute the loop body at all if the condition after the while fails during the first pass.

In the CHDK implementation of *uBasic,* loops can be nested to a depth of four.

5.3.7 Labels and GOTOs

The **goto** statement allows you to leave the linear execution and jump to another location in the script. While early *uBasic* versions only supported line numbers after the goto, recent *uBasic* versions support labels. This is much better because line numbers are subject to change when new lines are inserted into a script.

```
goto "restart"
```

Of course, such a label needs to be defined—and this is the syntax:

```
restart:
    print "script restarted"
```

While goto statements seem to be very convenient and flexible, they impose dangers, too. GOTOs are considered harmful in professional programming because programs tend to become incomprehensible and difficult to maintain if there are many of them jumping all over the place. So, if you can, use if ... then ... else ... endif, for ... next, or while ... wend instead of goto. In a tiny script, however, I can't see why a goto should do any harm.

5.3.8 Subroutines

Subroutines have a very similar syntax as GOTOs, but they can simplify programs instead of making them more difficult to read. A subroutine definition starts with a label and ends with a **return** statement:

```
:waitdisp
     do
        print "Continue: DISP"
        wait_click 3000
     until is_pressed "display"
return
```

This subroutine prompts the user to press the DISP key and then waits until this key is pressed. The advantage of defining such a piece of code in a subroutine is that it can be called from any location within the script by using the **gosub** statement:

```
gosub "waitdisp"
shoot
```

The return command within the subroutine will cause program control to return to the line after the gosub command and execute the instruction found there.

The CHDK *uBasic* implementation supports nested subroutines to a depth of 10.

5.3.9 Comments

Anything behind a **rem** command (including the rem command) is regarded as a comment and has no influence on processing.

```
gosub "waitdisp" rem waits on disp
```

Almost no influence, that is.

```
rem waits on disp
gosub "waitdisp"
```

will execute slower because the extra line attracts a delay of 10 milliseconds. So don't overdo it with comments in scripts. Comments need space, too, and space can become scarce for complex scripts. The CHDK sets an upper limit of 8 KB on each *uBasic* script.

5.3.10 Script structure

A script starts with a *header section* that defines the title and the parameters, as we saw in section 5.2. This is followed by the main body of the script. Subroutines are usually defined at the end of the script.

The main body typically consists of seven parts:

- The first section *informs the user* which script functions are triggered by pressing buttons.
- The second section makes sure that *no bad parameter values* are passed to the processing section. Usually, when a bad parameter value is detected, the default parameter value is assigned. In addition, you may want to print a message onto the display.
- The third section *initializes the variables* used in the processing section. Initial variable values may be fixed or may depend on parameter values. It may also be necessary to scale parameter values. For example, a parameter accepts values in centimeters but is fed into a *uBasic* command that expects millimeters. In this case, we would multiply the parameter value by 10.
- The fourth section saves the *initial camera state* values into variables so that the camera state can be restored at the end of the script.
- Then, the *main processing* section performs the task that was intended by the script.
- Finally, at the end of the script, the initial *camera state is restored*. If the script does not run to the end by itself (because it runs in a loop), you

should always offer a button to end script execution gracefully. Pressing the shutter button to interrupt a script should only be done in case of emergency, because it is not possible to restore the initial state.

- The final command in a script is the **end** command.

5.4 Lua primer

To the photographic community, *Lua* is known as the scripting language for *Adobe Lightroom*. So, if you have already developed scripts for *Lightroom,* you have a head start on CHDK scripting. Under the CHDK, *Lua* features more commands and functions than *uBasic,* is more platform-independent, and executes about 100 times faster. On the other hand, *Lua* is an advanced script language that you cannot master in one afternoon. Here, we will only introduce the basics of *Lua;* a description of the complete language would be beyond the scope of this book. If you are interested in *Lua* beyond CHDK scripting, we recommend reading the reference manual [Lua51Ref].

5.4.1 Variables

Unlike *uBasic,* variable names in *Lua* can be longer than a single character. This allows for an arbitrarily large number of different variables. A variable name must start with a letter or an underscore (_). However, variable names used as CHDK parameters in the script header (section 5.4.12) must consist of a single lower-case letter and can only accept numeric values. There is also an anonymous variable—typically used as a placeholder—with only the underscore (_) as its name.

Variables can be defined as global or local. Local variables are only available in the local context of the defining block (section 5.4.6) or function (section 5.4.9). Their definition starts with the keyword **local**, e.g.,

```
local speed = 250
```

In contrast to *uBasic,* variables are not restricted to numeric values; they can also contain strings, complex structures (tables), and functions (section 5.4.9). The initial value of a variable is **nil** (nothing). In the CHDK implementation of *Lua,* numeric values are restricted to integer values—floating-point arithmetic is *not* supported. This has implications for portability; the result of computations may differ depending on the platform where you execute the script (section 5.8).

5.4.2 Strings

Strings are enclosed in single (‘...’) or double (“...”) quotes, or in doubled square brackets ([[...]]). With this latter form, it is possible to define strings that stretch across multiple lines. Strings can contain most of the escape sequences used in other programming languages such as C or Java: \f for a form feed, \n for a new line, \r for the carriage return. Strings can be concatenated with the .. operator:

```
"CH"..'DK'
```

5.4.3 Tables

Tables are a core concept in *Lua*. Tables can contain an arbitrary number of elements—numbers, strings, functions, and other tables.

Table definitions are enclosed in curly brackets ({}). Anything (except nil) can be a table element. Table elements can be addressed through the element index. For instance:

```
disp_table = {'info', 'no_info',
     'off', 'electronic_viewfinder'}
print(disp_table[2])
```

would print ‘no_info’ because indexing starts at 1. The number of table elements can be obtained through the operator #:

```
print(#disp_table)
```

would print ‘4’.

Alternatively, table elements can be associated with explicit keys, in which case the table works as a dictionary:

```
reverse_disp_table = {info = 0,
             no_info = 1, off = 2,
             electronic_viewfinder = 3}
```

In this case we could address the elements by their key:

```
print(reverse_disp_table["info"])
```

would print ‘0’. The following dot notation is also possible:

```
print(reverse_disp_table.info])
```

Such dictionary tables should not be accessed via indices because no assumptions can be made about the order in which the elements are stored. Note that keys must not contain white space.

5.4.4 Assignments

Assignments look very similar to those in *uBasic,* except that the `let` command is not used:

```
a = b * c
```

Lua supports the same arithmetic operators as *uBasic,* plus the **^** operator for exponentiation.

Assigning values to table elements is possible, too. For example:

```
t[i] = [[multi-line
            string]]
```

Another difference is that lists of values can be assigned to lists of variables. For example:

```
a, b = 3, 4
```

assigns 3 to `a` and 4 to `b`. This form of assignment is often used for functions that return multiple values. If individual values are not needed, the anonymous variable "_" can be used:

```
_, b = 3, 4
```

Here, 4 is assigned to `b` and 3 is thrown away.

5.4.5 Output

The **`print()`** function can be used to create output on the display. Similar to *uBasic,* several parameters can be specified, but the semantics are different: individual values are separated by `TAB`, not by a single white space. The semicolon cannot be used to concatenate two strings. Instead, we use the string concatenation operator (**`..`**). For example:

```
print('a: '..a..'s')
```

would produce the output

```
a: 500s
```

if the variable `a` had the value of 500. Unlike *uBasic, Lua* allows the output of strings longer than 25 characters. Longer strings are wrapped into the next line.

The CHDK functions **cls()** and **print_screen()** work exactly the same as the equivalent commands in *uBasic*. There is one exception:

```
print_screen(false)
```

can be used to disable output to a log file.

5.4.6 Blocks

Blocks are statement sequences that are enclosed by **do ... end**. Their main purpose is to allow the definition of local variables that are only valid within the block. Blocks may be nested.

5.4.7 Conditional clauses

Similar to *uBasic,* conditional clauses are built with the **if ... then ... else ... end** construct. The difference is that the construct is not closed by a specific `endif` token but by the more generic `end` token. For example:

```
if a < 0 then
    b = 3
else
   b = 4
end
```

The `else` clause may be omitted, but in contrast to *uBasic,* it is not possible to omit the `end` token even for single line constructs:

```
if n < 0 then n = 0 end
```

Nested `if` statements can be simplified with the **elseif** token. Instead of:

```
if a < 0 then
    b = -1
else
    if a > 0 then
        b = 1
    else
        b = 0
    end
end
```

it is possible to write:

```
if a < 0 then
    b = -1
elseif a > 0 then
    b = 1
else
    b = 0
end
```

which is easier to read and to understand. *Lua* does not feature a `select` statement (section 5.3.5), but a sequence of `elseif` clauses can be used for the same purpose.

Relational operators allowed in *Lua* are:

== (equals) (*uBasic:* =)
> (greater than)
< (less than)
~= (not equal) (*uBasic:* <>)
<= (less than or equal)
>= (greater than or equal)

It is also possible to connect several conditions with the operators **and**, **or**, and **not**. Parentheses may be used to indicate which operators are executed first:

```
if (a = 1 or a = 2) and b < 0 then ...
```

5.4.8 Loops

Similar to *uBasic,* the **for**-loop is used for incrementing a counter from a start value to an end value and for executing the instructions within the loop for each counter value. For example:

```
for i = 1,n do
    sleep(10000)
    shoot()
end
```

takes `n` shots every 10 seconds.

By default, the increment value of the `for` construct is 1. By using a third value in the `for` clause, however, it is possible to specify a different increment value:

```
for i = 1,10,3 do
    print(i)
end
```

would print the values 1, 4, 7, 10.

A second form of the `for` statement—called the generic `for` statement—allows iteration over a series of values, such as the elements of a table. Using the built-in function **`ipairs()`**, we can write:

```
disp_table = {'info', 'no_info', 'off', 'electronic_viewfinder'}
for i,v in ipairs(disp_table) do
      print(i,v)
end
```

This results in the output:

```
1   info
2   no_info
3   off
4   electronic_viewfinder
```

The same is possible for dictionary tables that are accessible via keywords. Instead of `ipairs()`, the function **`pairs()`** is used in this case:

```
t = {info = 0, no_info = 1, off = 2, electronic_viewfinder = 3}
for key,value in pairs(t) do
     print(key,value)
end
```

The result of this script would be:

```
electronic_viewfinder    3
off                      2
info                     0
no_info                  1
```

Note that in this case the order in which the values are obtained cannot be predicted.

In addition to the various flavors of the `for` statement, *Lua* also features a **`while`** statement and a **`repeat`** statement. The latter replaces the `do...until` construct found in *uBasic*.

```
i = 1
while i <= n do
    sleep(10000)
    shoot()
      i = i + 1
end
```

does exactly the same as the first `for` loop shown at the beginning of this section.
The construct `repeat ... until` behaves quite similarly:

```
i = 1
repeat
    sleep 10000
    shoot
      i = i + 1
until i > n
```

The difference in the `while` construct, however, is that the loop body is executed at least once because the condition is checked at the end of the loop. The `while` construct may decide not to execute the loop body at all if the condition after the `while` fails during the first pass.

Both `while` and `repeat` loops can be aborted with the **`break`** statement, which always exits the innermost loop:

```
i = 1
while true do
      if i > n then break end
    sleep(10000)
    shoot()
      i = i + 1
end
```

Here we have used the Boolean value `true` in the `while` condition, which results in a loop that never stops. To break the loop, we check the inverse condition in the `if` statement and execute the `break` command if the condition holds.

The above control statements are fully sufficient to express any kind of control flow in a script—`goto` statements and labels are not really necessary. Consequently, *Lua* does not offer GOTO.

5.4.9 Functions

Instead of subroutines as in *uBasic, Lua* features more versatile functions. Each function consists of a function header, with a function name and parameters, and a function body. The whole function definition is enclosed by the pair **`function ... end`**. The list of parameters is enclosed in parentheses and separated by commas. Even parameterless functions use these parentheses with—aka—an empty list of parameters. For example:

```
function waitdisp()
   repeat
        print("Continue: DISP")
        wait_click(3000)
   until is_pressed("display")
end
```

In contrast to *uBasic,* functions can return one or several values. This is done with the **`return`** command followed by a single return value or by a comma-separated list of return values. The following function returns the time difference between the function start and a click on the specified button in milliseconds:

```
function stopwatch(k)
   t = get_tick_count()
   repeat
        wait_click(3000)
   until is_pressed(k)
   d = get_tick_count()-t
   return d
end
```

A special statement to invoke a function (like the `gosub` in *uBasic*) is not necessary—it is sufficient to simply invoke the function by its name, followed by the list of parameters enclosed in parentheses. For example:

```
d = stopwatch("display")
print(d)
```

A very powerful feature is the ability to assign a function definition to a variable. The function definition can then be passed as a parameter to other functions, stored in a table, or even returned as a result of another function. The following code assigns an anonymous function to the variable `hello`:

```
hello =
    function() print('hello') end
```

The function can later be invoked via

```
hello()
```

5.4.10 Error handling

Another big advantage of *Lua* over *uBasic* is the ability to run pieces of code in protected mode. This means that an error condition detected inside this code is not propagated to the surrounding code. Your script can gracefully handle this error and continue to run. The built-in function **pcall()** can execute other functions in protected mode. For example:

```
status,result = pcall(stopwatch,k)
```

As you can see, `pcall()` can return multiple values:

- If no error occurred, the first return value (`status`) has the value `true`, followed by the return values (if any) of the called function. In the above example, `result` would contain the result of function `stopwatch()`.
- If an error occurred, the first returned value is `false`, followed by the error message in the second return value.

In addition to the errors raised by *Lua* itself, errors can also raised by a script through the **error()** function. The first parameter of this function is an error message. The optional second parameter may contain an error level that indicates which additional information the *Lua* interpreter will add to the error message:

- **0:** no additional information.
- **1** (default): the location where the `error()` function was called.
- ***n:*** the error location *n* levels up in the caller hierarchy.

Lua also knows an **assert** instruction, which is usually used to test the conditions under which a piece of code is allowed to run.

```
assert(cond, msg)
```

If the expression `cond` results in `false`, an error is raised with the content of the parameter `msg` as an error message. The code following the `assert`

instruction can therefore be sure that the condition holds. Using `assert` frequently can considerably improve the safety and robustness of a script.

5.4.11 Comments

Anything behind a **--** token (including the `--` token) is regarded as a comment and has no influence on processing. For example:

```
wait_click(3000) -- wait 3 secs
```

Multi-line comments are simply written as a multi-line string behind the comment token:

```
--[[This comment
stretches across multiple lines]]
```

5.4.12 Script structure

Just like *uBasic* CHDK scripts, *Lua* CHDK scripts start with a *header section* defining title and script parameters. This is followed by the main body of the script as outlined in section 5.3.10. The title and parameter section must be provided in the form of a multi-line comment containing the title and parameter definitions as known from *uBasic*.

```
--[[
rem 22-Sep-2009 by bdaum
@title Countdown
@param t ticks(sec)
@default t 10
]]
```

Comments within such a block can be written with the *uBasic* syntax (`rem`).

5.4.13 Standard Libraries

Unlike *uBasic, Lua* features a library concept so that new functionality can be added to the *Lua* core without much effort. Libraries are sets of predefined *Lua* functions that you can invoke in your program after you have loaded the library. In addition, a library can immediately execute statements during the loading process. It is even possible for a library to return values to the loading script by executing a `return` command.

Some standard libraries are already included in *Lua,* such as:

- Basic library for some *core functions*
- *Input and output* library
- *Operating system* facilities library
- *String manipulation* library
- *Mathematical functions* library. The CHDK version of this library differs substantially from the standard version because all functions dealing with floating point numbers have been removed (section 5.4.1).

Other libraries can easily be added to *Lua* by placing them into the folder `/CHDK/LUALIB`. Before such a library can be accessed in a script, it must be loaded. This is done with the statement:

```
require("library_name")
```

where the library name is written without the suffix ".lua". It is possible to assign the result of function `require()` to a variable. Often libraries construct a table containing all public functions and values defined in the library, and return this table to the loader. The loader may then retrieve these functions and values. For example:

```
props = require("propcase")
id = props.DISPLAY_MODE
```

Here the library `propcase.lua` has returned the table `props` containing a value under key `DISPLAY_MODE`. This technique helps avoid name clashes when several libraries are used at the same time.

The following libraries are automatically loaded:

Lua core functions

The core library implements functions such as the already discussed `pcall()`, `error()`, or `print()`. The following functions can also be useful:

`dofile(fname)`	Loads the specified file as a new *Lua* script and executes it.
`tonumber(p,b)`	Converts `p` to a number to the specified base `b`. If `b` is omitted, a base of 10 (the decimal system) is assumed.
`tostring(p)`	Converts the parameter `p` into a string.
`type(p)`	Returns the type of `p`. Possible return values are: "nil", "number", "string", "boolean, "table", "function", "thread", and "userdata".

IO functions

The *Lua* IO library provides access to file input and output. Using these functions, it is possible to read and write files located on the memory card. We will only give a short overview of the most important IO functions. Examples for the application of these functions are found in section 5.7.1.

```
file, msg, no = io.open(filename, mode)
```

opens a file with the specified name and in the specified mode. A "b" appended to the mode strings indicates a binary file:

"r"	Read mode (the default if the mode parameter is omitted).
"w"	Write mode. A new file is created, and existing files are overwritten.
"a"	Append mode. Appends to the end of an existing file.

In case of an error, the return variable file is nil and an error message is given in msg, an error code in no.

```
ret, msg, no = io.close(file)
ret, msg, no = file:close()
```

Closes the specified file. In case of an error, ret is nil, and an error message is given in msg, an error code in no.

```
ret, msg, no = io.flush()
```

Performs all outstanding buffered write operations and makes sure that all written data is physically stored.

```
ret, msg, no = file:lines()
```

Creates an iterator over the lines in a text file. It can be used in the following way:

```
for line in file:lines()
  do ... end
```

Here, the variable line will subsequently contain each line in the specified file. If no more lines are available, nil will be returned.

```
value1,... = file:read(format1,...)
```

For each of the specified formats, `read()` will return a numeric or string value read from the input file. If the specified format cannot be satisfied, `nil` is returned instead. The following formats are available:

`"*n"`	Reads a number.
`"*a"`	Reads the whole file, starting at the current position. If the current position is at the end of the file, `nil` is returned.
`"*l"`	Reads the next line. `nil` is returned at the end of the file.
number	Reads a string with up to the specified number of characters. `nil` is returned at the end of the file.

```
pos, msg = file:seek(mode, offset)
```

positions the write and read pointers to the specified offset in the file, beginning at the origin specified in `mode`. `offset` can be omitted and defaults to 0 in that case. In case of an error, `pos` is `nil`, and an error message is given in `msg`.

`"set"`	Starts at the beginning of the file.
`"cur"`	Starts at the current position. This is the default when the mode parameter is omitted.
`"end"`	Starts at the end of the file and subtracts the offset.

Example:

```
function fsize(file)
  local current_pos = file:seek()
  local size = file:seek("end")
  file:seek("set", current_pos)
  return size
end
```

This function determines the size of a file. It first saves the current position into the local variable `current_pos`, and then positions to the file end and stores that position in the variable `size`. As a good citizen, it restores the original position and then returns `size` as result.

```
ret,msg,no = file:write(value1,...)
```

writes the values passed in the parameters to the specified file. Only numeric or string values are allowed as parameters. In case of an error, `ret` is `nil`, and an error message is given in `msg`, an error code in `no`.

Operating system functions

Only some of the standard *Lua* OS functions are supported in the CHDK.

```
r, msg = os.remove(filename)
```

deletes a file or directory with the specified name. If it succeeds, `true` is returned. If it fails, `nil` is returned, followed by an error message. The function will fail on an attempt to delete a directory that is not empty..

```
r,msg = os.rename(oldname,newname)
```

renames the file or directory specified in `oldname` to the name given in `newname`. If it succeeds, `true` is returned. If it fails, `nil` is returned, followed by an error message.

Caution! Attempting to rename a *nonempty directory* may result in file system corruption! Attempting to *move a file to another directory* by renaming it can lead to unpredictable results. The `rename()` function may produce unpredictable results if the *target name already exists*.

```
t = os.time(table)
```

returns the time specified in `table` as a single number (number of seconds since January 1, 1970). The table has the following keywords: “year”, “month”, “day”, “hour”, “min”, “sec”, “dst”, “wday”. “dst” represents a Boolean value that is `true` for daylight savings time; “wday” represents the day of the week (Sunday = 1).

If the parameter `table` is missing, the function delivers the current date and time as a number:

```
s = os.date(format, time)
```

The `date()` function works the other way round. The format string specifies how a numeric date value will be converted into a string. If the second parameter is missing, the current date is used. The format string can contain short control sequences (‘%’ followed by a letter) that will be replaced by year, month, day, etc. The following control sequences are possible:

%a	Abbreviated weekday name (e.g., Thu)
%A	Full weekday name (e.g., Thursday)
%b	Abbreviated month name (e.g., Sep)
%B	Full month name (e.g., September)
%c	Date and time (e.g., 09/24/09 23:57:10)
%d	Day of the month (e.g., 24)
%H	Hour, 24-hour clock (e.g., 23)
%I	Hour, 12-hour clock (e.g., 11)
%M	Minute (e.g., 57)
%m	Month (01 = January, e.g., 09)
%p	Either "am" or "pm" (e.g., pm)
%S	Second (e.g., 10)
%w	Weekday (0 = Sunday, e.g., 4)
%x	Date (e.g., 09/24/09)
%X	Time (e.g., 23:57:10)
%Y	Full year (e.g., 2009)
%y	Two-digit year (e.g., 09)
%%	The character '%' when used as a literal

For example:

```
s = os.date("today, %B %d, %Y")
```

returns

```
"today, September 24, 2009"
```

A special case is the format string `'*t'`. In this case, the `date()` function returns date and time in the form of a table, as seen in the discussion of the `time()` function.

In addition to the above functions, the CHDK implementation provides some more functions that are not part of the standard *Lua* OS library (which implements only those OS functions that are part of ANSI C). Consequently, scripts utilizing those functions cannot be tested on another platform such as a PC, except with an appropriate OS extension library (section 5.8).

```
r, msg, no = os.mkdir(dirname)
```

creates a new directory with a specified path name such as "A:/CHDK/WORK". Please note that the path name must not end with a slash. If the function succeeds, true is returned. If it fails, nil is returned, followed by an error message and an error number.

```
t, msg, no = os.stat(filename)
```

delivers a table t containing information about the specified file.

Key	Value
dev	Device number
mode	Meaning unclear
size	Size in bytes
atime	Last access time
mtime	Last modification
ctime	Last status change
blksize	Block size in bytes
blocks	Number of blocks in the file
attrib	Bit mask of DOS attributes
is_dir	true for a directory
is_file	true for a file

If the function fails, nil is returned, followed by an error message and an error number.

```
r, msg, no =
     os.utime(filename,atime,mtime)
```

sets the time of last access and the time of last modification for the specified file. If atime or mtime is omitted or nil, the current time is used. If the function succeeds, true is returned. If it fails, nil is returned, followed by an error message and an error number.

```
t, msg, no =
        os.listdir(dirpath,showall)
```

lists the contents of the specified directory. The result is returned as a table of file names. If the function fails, `nil` is returned, followed by an error message and an error number. If the optional parameter `showall` is `true`, the list will contain the entries "`.`", "`..`", and deleted entries, too.

String manipulation functions

The CHDK implements the *Lua* string manipulation library completely. The library comes with some powerful functions:

`l = string.len(s)`

Returns the character length of a string. The same is achieved with the operator `#`:

```
l = #s
```

`t = string.rep(s,n)`

Returns a string where string `s` is repeated `n` times.

`t = string.upper(s)`

Returns string `s` converted to upper case.

`t = string.lower(s)`

Returns string `s` converted to lower case.

`t = string.sub(s, from, to)`

Returns a substring of `s` starting at position `from` and ending at position `to`. Indexes start at 1; negative indexes are counted from the end of the string.

`t = string.char(n,...)`

Converts the integer parameters to characters and concatenates them to a string.

`n = string.char(s, i)`

Converts the i-th character of s into an integer. Indexes start at 1; negative indexes are counted from the end of the string.

`t = string.format(f, n1,...)`

Formats the content of the parameters `n1,...` with the help of format description `f`. The formatting rules are quite similar to that of the `printf()` statement of ANSI C.

For example

```
tag, title = "b", "chdk"
string.format("<%s>%s</%s>", tag, title, tag)
```

returns

```
<b>chdk</b>
```

Often used format characters are the following:

`%d %i`	Decimal signed integer
`%o`	Octal integer
`%x %X`	Hex integer
`%u`	Unsigned integer
`%c`	Character

%s	String
%q	Quoted string. The string is formatted so that it can safely be read back by *Lua*.
%%	%. No argument expected.

i,j = string.find(s1, s2, k)

Searches for the substring s2 in string s1 and returns the indices for the first and the last character of s2. If s2 is not a substring of s1, the function returns nil. The parameter k is optional and may specify the start index for the search.

t,n = string.gsub(s1,s2,s3,k)

Replaces all occurrences of string s2 in string s1 with string s3. Returns the resulting string and the replacement count. The optional parameter k can limit the number of replacements.

The functions find() and gsub() accept patterns, too—strings with interspersed control sequences. Most control sequences start with "%" followed by a character, optionally followed by a modifier character (see below). These control sequences describe character classes such as "letters" or "digits".

.	All characters
%a	Letters
%c	Control characters such as new line or form feed
%d	Digits
%l	Lowercase letters
%p	Punctuation characters
%s	Whitespace characters
%u	Uppercase letters
%w	Alphanumeric characters
%x	Hexadecimal digits
%z	The character "0"
%	Escape character. % followed by another character represents the character itself. Escaping is particularly necessary for the characters ( ) . % + - * ? [^ $. For example, "%%" represents "%".

Using uppercase control characters after the "%" negates the character class identified by the corresponding lowercase character. For example, %S stands for anything except white space.

You can define your own character sets by enclosing the characters belonging to the set within brackets:

[+–] for example, means '+' or '–'.

A modifier character describing how often a character of that class is expected can follow a character class sequence:

none	Exactly one occurrence
+	At least one occurrence
*	Any number of occurrences; works greedily (tries to consume longest sequence possible to succeed)
-	Any number of occurrences; works non-greedily (tries to consume shortest sequence possible to succeed)
?	Optional (0 or 1 occurrence)

Command sequences that match a character sequence can be captured, meaning that the matching character string will be accessible. A capture is achieved by putting a command sequence into parentheses. The matching character sequence is then delivered as an extra result by the `find()` function. For example:

```
_, _, m, d, y = string.find(
  "9/25/2009", "(%d+)/(%d+)/(%d+)")
```

assigns 9 to `m`, 25 to `d`, 2009 to `y`. Each capture `(%d+)` tries to find one or more digits until the next slash or the end of the string, and it delivers the matching substring as a result.

Capture results can be used in the pattern itself or in the substitution string of the `gsub()` function. They are symbolized by the control sequences `%1`, `%2`, ... representing the result of the first capture, the second capture, and so on. For example:

```
string.gsub("log.txt",
       "(%w+)%.(%w+)", "%1_1.%2")
```

would result in "log_1.txt".

Table manipulation functions

Because tables are an essential construct in *Lua,* there is also a standard library with functions for conveniently manipulating them.

`s = table.concat(table,sep,i,j)`

Concatenates the table elements `i..j` into a string, each separated by separator `sep`—provided all elements are strings or numbers. `sep`, if omitted, defaults to the empty string. `i` and `j` default, if omitted, to 1 resp. the length of the table.

table.insert (table, pos, value)

Inserts value at the specified position into the table, shifting existing elements at and behind that location one position towards the end of the table. If pos is omitted, the value is appended at the end of the table.

m = table.maxn (table)

Returns the maximum positive index of the table. If no positive index exists, 0 is returned.

e = table.remove (table [, pos])

Removes the element at the specified position from the table and returns it. If pos is not supplied, the very last element in the table is removed.

table.sort (table, comp)

Sorts the elements in the table. Optionally, a function can be supplied in comp defining the sort order. The function receives two table elements as parameters and must return true if the first element is considered smaller than the second element.

Example:

```
table.sort(table,
    function(a,b)
        return a > b
    end
)
```

will sort the table in the opposite order because it returns true when the second parameter is smaller than the first.

Mathematical functions

Because the CHDK implementation of *Lua* only supports integer numbers, mathematical functions in the *Lua* mathematical library, such as sin() or log(), would make no sense. The mathematical library therefore boils down to:

math.max()	Returns the maximum of its arguments
math.min()	Returns the minimum of its arguments
math.pow(x,y) or x^y	Raises the first parameter to the power of the second parameter and returns the result
math.random(x,y)	Generates random numbers between x and y. If x is omitted, a lower limit of 0 is assumed.
math.randomseed()	Sets a start value for the pseudo random generator. A typical way to initialize the random generator is randomseed(os.time()).

5.5 CHDK commands

In sections 5.3 and 5.4, we looked at the language features of *uBasic* and *Lua*. None of these language features are able to invoke specific camera functions. For that purpose, the CHDK implementations of *uBasic* and *Lua* provide extra commands enriching both languages. The names of the commands are the same in both languages. The main difference is that in *Lua,* parameters are enclosed in parentheses and separated by commas, whereas the CHDK command is implemented as a function. In *uBasic,* the parameters are simply listed behind the command name and separated by blanks. While values can be returned through parameters in *uBasic, Lua* only allows returning values as a function result. For example:

```
is_key r "right"    uBasic
r = is_key("right")  Lua
```

Some commands return Boolean values (`true` or `false`). Unlike *Lua, uBasic* does not have a Boolean datatype. Those values are therefore returned to *uBasic* as 0 (`false`) or 1 (`true`).

5.5.1 Button-related commands:

`press "button-name"` *uBasic*
`press("button-name")` *Lua*

A button is pressed and held down. Normally, this command is followed later by a `release` command for the same button name.

`release "button-name"` *uBasic*
`release("button-name")` *Lua*

A button is released. If the button was not pressed, the command does not have any effect.

`click "button-name"` *uBasic*
`click("button-name")` *Lua*

A button is pressed and immediately released.

`shoot` *uBasic*
`shoot()` *Lua*

Similar to `click "shoot_full"`. In contrast to the `click` command, the `shoot` command will, however, perform all the automatic actions such as focusing, exposure control, and flash setup before releasing the shutter.

wait_click t *uBasic*
wait_click(t) *Lua*
wait_click() *Lua*

Waits for a key click. The optional parameter `t` can specify a time-out value in milliseconds. If `t` is not specified or has the value 0, no time-out will occur.

is_pressed r "button-name" *uBasic*
r = is_pressed "button-name" *uBasic*
r = is_pressed("button-name") *Lua*

Typically used after a `wait_click` command for determining the button clicked. If the clicked button matches the specified name, a value of 1 resp. `true` is returned. By using the pseudo button name `"no_key"`, it is even possible to check for a time-out.

is_key r "button-name" *uBasic*
r = is_key "button-name" *uBasic*
r = is_key("button-name") *Lua*

Determines if the specified button is pressed. If the pressed button matches the specified name, a value of 1 resp. `true` is returned. Typically used standalone without `wait_click`.

wheel_right *uBasic*
wheel_left *uBasic*
wheel_right() *Lua*
wheel_left() *Lua*

Only for PowerShot G7 and SX100IS. Turns the multi-control wheel one stop to the right or to the left.

r = get_video_button *uBasic*
r = get_video_button() *Lua*

Returns 1 resp. `true` if the camera has a video button (S-series, TX1).

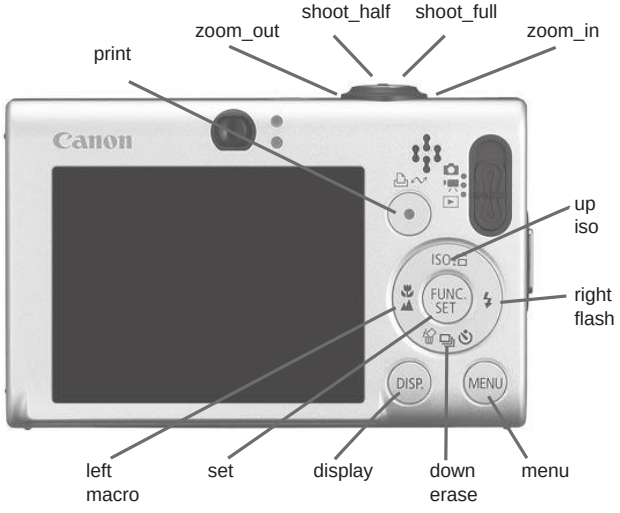


Figure 5-2
Button names that can be used with the commands `press`, `release`, and `click`. Additional buttons on some cameras are `mf` (manual focus), `video`, `timer`, `expo_corr` (exposure correction), and `fe` (microphone). On S-series, `print` stands for the **shortcut** button, `erase` stands for the **func** button. Pseudo button names are `no_key` for a time-out and `remote` for the USB remote control.

5.5.2 Exposure-related commands

As already explained in section 4.2.7, the exposure system of Canon cameras is engineered along the APEX system. All entities *(Av, Tv, Ev, Sv,* and *Bv)* use a logarithmic scale and are divided into 1/3 f-stop. These units are identified by an index number. Internally, the camera uses an even finer scale: 32 units per 1/3 f-stop (96 units per f-stop). These finer units can also be accessed via CHDK functions. Please see Table 5-1 for *Tv* index and *Tv96* numbers and Table 5-2 for *Av* index and *Av96* numbers.

When not using CHDK *Overrides* (section 4.3.1), the camera will determine these values from the measured scene brightness and the selected exposure program. When using CHDK *Overrides,* however, the user can set each of those entities to a fixed value. Depending on the camera, exposure control then works in aperture priority mode or exposure time priority mode. In the case of cameras without diaphragms, it leaves everything to the user or to a script; because all of the above entities can be retrieved and most (except *Bv*) can be set via script commands, it is possible to organize exposure control through scripts (section 5.7.4).

Let's look at the CHDK commands that are available for exposure control.

Exposure time (Tv)

r = get_nd_present	*uBasic*
r = get_nd_present()	*Lua*

Returns info about the neutral density (ND) filter: 0 = filter present, 1 = no built-in ND filter, 2 = camera has both a diaphragm and an ND filter.

get_tv x	*uBasic*
x = get_tv()	*Lua*
set_tv x	*uBasic*
set_tv(x)	*Lua*

Retrieves and sets the real exposure time *(Tv)* index. `x = 0` stands for an exposure time of 1 second. Negative values indicate longer exposure times, positive values indicate shorter times (3 units per f-stop).

get_tv96 x	*uBasic*
x = get_tv96()	*Lua*
set_tv96 x	*uBasic*
set_tv96(x)	*Lua*

Retrieves and sets the internal real *Tv* value (96 units per f-stop).

get_user_tv_id x	*uBasic*
x = get_user_tv_id()	*Lua*
set_user_tv_id x	*uBasic*
set_user_tv_id(x)	*Lua*

Retrieves and sets the index of the exposure time set by the user (3 units per f-stop).

get_user_tv96 x *uBasic*

x = get_user_tv96() *Lua*

set_user_tv96 x *uBasic*

set_user_tv96(x) *Lua*

Retrieves and sets the value of the internal exposure time set by the user (96 units per f-stop).

set_tv_rel x *uBasic*

set_tv_rel(x) *Lua*

Modifies the current exposure time. A negative x makes the exposure time longer, a positive x makes it shorter (3 units per f-stop).

set_user_tv_by_id_rel x *uBasic*

set_user_tv_by_id_rel(x) *Lua*

Modifies the exposure time set by the user (3 units per f-stop).

set_tv96_direct x *uBasic*

set_tv96_direct(x) *Lua*

Sets the exposure time value with 96 units per f-stop.

Table 5-1
Exposure index vs. exposure time

Index	Tv96	Speed	Index	Tv96	Speed
–12	–384	*15"*	18	576	*1/60*
–11	–352	*13"*	19	608	*1/80*
–10	–320	*10"*	20	640	*1/100*
–9	–288	*8"*	21	672	*1/125*
–8	–256	*6"*	22	704	*1/160*
–7	–224	*5"*	23	736	*1/200*
–6	–192	*4"*	24	768	*1/250*
–5	–160	*3"2*	25	800	*1/320*
–4	–128	*2"5*	26	832	*1/400*
–3	–96	*2"*	27	864	*1/500*
–2	–64	*1"6*	28	896	*1/640*
–1	–32	*1"3*	29	928	*1/800*
0	0	*1"*	30	960	*1/1000*
1	32	*0"8*	31	992	*1/1250*
2	64	*0"6*	32	1024	*1/1600*

Table 5-1 Continued

Index	Tv96	Speed	Index	Tv96	Speed
3	96	*0"5*	33	1056	*1/2000*
4	128	*0"4*	34	1088	*1/2500*
5	160	*0"3*	35	1120	*1/3200*
6	192	*1/4*	36	1152	*1/4000*
7	224	*1/5*	37	1184	*1/5000*
8	256	*1/6*	38	1216	*1/6400*
9	288	*1/8*	39	1248	*1/8000*
10	320	*1/10*	40	1280	*1/10000*
11	352	*1/13*	41	1312	*1/12800*
12	384	*1/15*	42	1344	*1/16000*
13	416	*1/20*	43	1376	*1/20000*
14	448	*1/25*	44	1408	*1/25000*
15	480	*1/30*	45	1440	*1/32000*
16	512	*1/40*	46	1472	*1/40000*
17	544	*1/50*	47	1504	*1/50000*

Aperture (Av)

Note: Not surprisingly, setting aperture values on cameras without diaphragms has no effect.

get_av x	*uBasic*
x = get_av()	*Lua*
set_av x	*uBasic*
set_av(x)	*Lua*

Retrieves and sets the real aperture *(Av)* index (Table 5-2). `x = 0` stands for an aperture of *ƒ/1*. Larger values stand for smaller apertures (3 units per f-stop).

get_av96 x	*uBasic*
x = get_av96()	*Lua*
set_av96 x	*uBasic*
set_av96(x)	*Lua*

Retrieves and sets the real aperture internal value (96 units per f-stop).

get_user_av_id x *uBasic*
x = get_user_av_id() *Lua*
set_user_av_by_id x *uBasic*
set_user_av_by_id(x) *Lua*

Retrieves and sets the index of the aperture index set by the user (3 units per f-stop).

get_user_av96 x *uBasic*
x = get_user_av96() *Lua*
set_user_av96 x *uBasic*
set_user_av96(x) *Lua*

Retrieves and sets the internal value of the aperture set by the user (96 units per f-stop).

set_av_rel x *uBasic*
set_av_rel(x) *Lua*

Modifies the real aperture relative to the current aperture. A negative x makes the aperture wider, a positive x makes it smaller (3 units per f-stop).

set_user_av_by_id_rel x *uBasic*
set_user_av_by_id_rel(x) *Lua*

Modifies the aperture set by the user (3 units per f-stop).

set_nd_filter f *uBasic*
set_nd_filter(f) *Lua*

Sets the neutral density filter: 0 = off, 1 = in, 2 = out. This command has no effect on cameras without an ND filter.

Table 5-2
Aperture index vs. aperture

Index	Av96	Aperture
9	288	*f/2.8*
10	320	*f/3.2*
11	352	*f/3.5*
12	384	*f/4.0*
13	416	*f/4.5*
14	448	*f/5.0*
15	480	*f/5.6*
16	512	*f/6.3*
17	544	*f/7.1*
18	576	*f/8.0*

Sensor speed (Sv)

`get_iso x`	*uBasic*
`x = get_iso()`	*Lua*
`set_iso x`	*uBasic*
`set_iso(x)`	*Lua*

Retrieves and sets the ISO speed *(Sv)* index (Table 5-3). The index values > 0 are mapped to ISO values of 50 and above. The index values –1 and 0 have a special meaning and stand for *HiISO* and *AutoISO*.

`get_iso_mode x`	*uBasic*
`x = get_iso_mode()`	*Lua*
`set_iso_mode x`	*uBasic*
`set_iso_mode(x)`	*Lua*

Retrieves and sets the ISO mode
(–1 = *HiISO,* 0 = *AutoISO,* 50, 100, 200, 400, 800).

`get_iso_real x`	*uBasic*
`x = get_iso_real()`	*Lua*
`set_iso_real x`	*uBasic*
`set_iso_real(x)`	*Lua*

Retrieves and sets the real ISO value. This is the real value used to control the sensor.

`get_iso_market x`	*uBasic*
`get_iso_market(x)`	*Lua*

Retrieves the "marketing" ISO value—the value shown on the display. This value is higher than the true ISO value (section 4.3.1).

`get_sv96 x`	*uBasic*
`x = get_sv96()`	*Lua*
`set_sv96 x`	*uBasic*
`set_sv96(x)`	*Lua*

Retrieves and sets the internal sensor speed *(Sv)* with 96 units per stop.

Table 5-3
ISO index vs. ISO value

ISO Index	Sv96	ISO mode
–1	–	*HiISO*
0	–	*AutoISO*
1	*384*	*50*
2	*480*	*100*
3	*576*	*200*
4	*672*	*400*

Table 5-3 Continued

ISO Index	Sv96	ISO mode
5	*768*	*800*
6	*864*	*1600*
7	*960*	*3200*
8	*1056*	*6400*

Light values (Bv, Ev)

get_bv96 x *uBasic*
x = get_bv96() *Lua*
Retrieves the scene brightness *(Bv)* with 96 units per stop.

get_ev x *uBasic*
x = get_ev() *Lua*
set_ev x *uBasic*
set_ev(x) *Lua*
Gets and sets the exposure value (Ev) with 96 units per stop.

5.5.3 Focus-related commands

These commands allow reading focus-related values such as subject distance, depth of field, and so on. They also allow setting the subject distance to a fixed value and then locking the autofocus system. On cameras with an explicit manual focus mode, the camera must be switched manually to that mode before the focus can be set via a script command. On cameras without an explicit manual focus mode, no action is required.

get_focus_mode x *uBasic*
x = get_focus_mode *uBasic*
x = get_focus_mode() *Lua*
Returns the focus mode: 0 = auto, 1 = manual.

get_focus d *uBasic*
d = get_focus *uBasic*
d = get_focus() *Lua*
set_focus d *uBasic*
set_focus(d) *Lua*
Retrieves and sets the focus distance in millimeters. The value of –1 resp. 65535 indicates an infinite distance.

get_dof x	*uBasic*
x = get_dof	*uBasic*
x = get_dof()	*Lua*

Returns the *Depth of Field* (DOF) in millimeters.

get_far_limit d	*uBasic*
d = get_far_limit	*uBasic*
d = get_far_limit()	*Lua*
get_near_limit d	*uBasic*
d = get_near_limit	*uBasic*
d = get_near_limit()	*Lua*

Sets the far and near limit (in millimeters) of the distance range with acceptable sharpness. The value of –1 resp. 65535 indicates an infinite distance, but also a distance outside the current camera setting (macro, normal).

get_hyp_dist d	*uBasic*
d = get_hyp_dist	*uBasic*
d = get_hyp_dist()	*Lua*

Returns the hyperfocal distance in millimeters. The hyperfocal distance is the closest distance to which the lens can be focused while keeping objects at infinity sharp. All objects within the range of half of the hyperfocal distance and infinity will be sharp.

set_aflock a	*uBasic*
set_aflock(a)	*Lua*

Sets autofocus lock: 0 = off, 1 = on. Especially useful for time series where refocusing is not desired. When the AF lock is set, the AF LED is switched off, too.

5.5.4 Zoom-related commands

s = get_zoom_steps	*uBasic*
s = get_zoom_steps()	*Lua*

Returns the maximum number of zoom steps. This may differ among camera models.

get_zoom z	*uBasic*
z = get_zoom	*uBasic*
z = get_zoom()	*Lua*
set_zoom z	*uBasic*
set_zoom(z)	*Lua*

Retrieves and sets the current zoom step.

Note: It is recommended to set the zoom speed explicitly with command `set_zoom_speed` (see below) before using the `set_zoom` command. Improper use of the `set_zoom` command can later result in the message "assert failed—game over!" and an immediate camera shutdown.

A safer method to change zoom levels is to apply simulated clicks to the `zoom_in` and `zoom_out` buttons (section 5.5.1).

set_zoom_rel d *uBasic*

set_zoom_rel(d) *Lua*

Sets increments or decrements of the current zoom step.

set_zoom_speed s *uBasic*

set_zoom_speed(s) *Lua*

Sets the zoom speed between 5 and 100 percent. This does nothing for A-series cameras.

5.5.5 Flash-related commands

f = get_flash_mode *uBasic*

f = get_flash_mode() *Lua*

Returns the flash mode: 0 = auto, 1 = on, 2 = off.

r = get_flash_ready *uBasic*

r = get_flash_ready() *Lua*

Indicates that the flash is ready with a value of 1/`true`.

5.5.6 Image-related commands

get_quality q *uBasic*

q = get_quality() *Lua*

Retrieves and sets the current image quality setting (0 = Superfine, 1 = Fine, 2 = Normal).

get_resolution q *uBasic*

q = get_resolution() *Lua*

Retrieves and sets the current image resolution (0 = L, 1 = M1, 2 = M2, 3 = M3, 4 = S, 5 = RAW (on G9), 6 = Postcard, 8 = W).

o = get_orientation_sensor *uBasic*

o = get_orientation_sensor() *Lua*

Returns the camera orientation (portrait, landscape) in degrees.

5.5.7 Time-related commands

sleep t *uBasic*

sleep(t) *Lua*

Specifies how long the script will wait in milliseconds. The timer resolution depends on the camera model and is in the range between 10–30 milliseconds.

s = get_day_seconds *uBasic*

s = get_day_seconds() *Lua*

Returns the number of seconds since midnight.

t = get_tick_count *uBasic*

t = get_tick_count() *Lua*

Returns the number of milliseconds since the camera was switched on.

t = get_time u *uBasic*

u = 0 returns second, u = 1 minute, u = 2 hour, u = 3 day, u = 4 month, u = 5 year.

t = get_time(u) *Lua*

u = 's' returns second, u = 'm' minute, u = 'h' hour, u = 'D' day, u = 'M' month, u = 'Y' year.

5.5.8 Display-related commands

x = get_display_mode *uBasic*

x = get_display_mode() *Lua*

Returns the display mode: 0 = info, 1 = no info, 2 = off, 3 = electronic viewfinder.

shot_histo_enable x *uBasic*

shot_histo_enable(x) *Lua*

Switches the histogram on or off: 0 = off, 1 = on.

get_histo_range s,h,p *uBasic*

p = get_histo_range(s,h) *Lua*

Retrieves histogram data from the previous shot. A shadow value (0–1023) is specified in the variable `s`; a highlight value (s–1023) is specified in the variable `h`. The value returned in variable `p` is the percent of pixels within that brightness range. Can be used for sophisticated exposure control.

md_detect_motion ... *uBasic*

h = md_detect_motion(...) *Lua*

Detects motion (section 5.7.3)

n = md_get_cell_diff *uBasic*

n = md_get_cell_diff() *Lua*

Allows analyzing detected motion in more detail (section 5.7.3).

playsound s *uBasic*

playsound(s) *Lua*

Plays a sound signal. The variable s specifies the signal to be played: 0 = power on, 1 = shutter, 2 = click, 3 = timer, 4 = short, 5 = AF, 6 = error, 7 = long.

Currently, there is no way to play a voice note.

set_backlight b *uBasic*

set_backlight(b) *Lua*

Switches the display backlight on or off: 0 = off, 1 = on

set_led a,b,c *uBasic*

set_led(a,b,c) *Lua*

Switches the LED on or off. Parameter a identifies the LED: 4 = green, 5 = yellow, 6 = power (not for all cameras), 7 = orange, 8 = blue, 9 = AF light, 10 = timer. Parameter b sets the state: 0 = off, 1 = on. The optional parameter c controls the brightness (0–200). This works only for the blue LED and not for all cameras.

5.5.9 Image management commands

r = get_raw_nr *uBasic*

r = get_raw_nr() *Lua*

set_raw_nr r *uBasic*

set_raw_nr(r) *Lua*

Retrieves and sets noise reduction for RAW images *(Dark Frame Subtraction)*: 0 = auto, 1 = off, 2 = on.

r = get_raw *uBasic*

r = get_raw() *Lua*

set_raw r *uBasic*

set_raw(r) *Lua*

Retrieves and sets the state of the RAW output function: 0 = off, 1 = on.

s = get_disk_size *uBasic*

s = get_disk_size() *Lua*

Returns the size of the memory card in kilobytes.

s = get_free_disk_space *uBasic*

s = get_free_disk_space() *Lua*

Returns the free space of the memory card in kilobytes.

c = get_exp_count *uBasic*

c = get_exp_count() *Lua*

Returns the number of images taken since the camera was switched on.

c = get_jpg_count *uBasic*

c = get_jpg_count() *Lua*

Estimates the number of JPG images that still can be stored on the memory card.

c = get_raw_count *uBasic*

c = get_raw_count() *Lua*

Estimates the number of RAW images that still can be stored on the memory card.

set_raw_develop("filename") *(Lua only)*

Develops the specified RAW file into JPEG on the next shot (section 4.5.5). If the parameter is omitted or `nil`, the pending development task is canceled. Development will be performed under the camera settings at the time of development.

set_curve_state(n) *(Lua only)*

Sets the development curve state for post-processing (section 4.3.8): 0 = None, 1 = Custom, 2 = +1EV, 3 = +2EV, 4 = AutoDR.

raw_merge_start(operation) *(Lua only)*

Starts a merging process for RAW files (section 4.5.6). Operation: 0 = sum, 1 = average. The following `raw_merge_add()` commands will contribute to the result.

raw_merge_add("filename") *(Lua only)*

Adds a single picture to a merged RAW image. Typically, this technique is used to reduce noise by averaging multiple shots (section 4.5.6).

raw_merge_end() *(LUA only)*

Completes the merge operation.

5.5.10 Camera state

x = get_platform_id *uBasic*

x = get_platform_id() *Lua*

Returns a numeric value identifying the camera model.

b = get_buildinfo() *(Lua only)*

Returns a table with information about the CHDK build. Table fields can be addressed with the following keys: `platform`, `platsub`, `version`, `build_number`, `build_date`, `build_time`.

x = autostarted *uBasic*

x = autostarted() *Lua*

Assigns a value of 1 resp. `true` to variable `x` if the script was started using the *Autostart* function (section 5.1).

a = get_autostart *uBasic*

a = get_autostart() *Lua*

set_autostart a *uBasic*

set_autostart(a) *Lua*

Retrieves or sets the state of the autostart function (section 5.1).

0 = autostart off. The user always starts scripts. **1 = autostart on**. The next time the camera is switched on, the previously selected script will be executed.

2 = autostart once. The next time the camera is switched on, the previously selected script will be executed. The autostart state is then set back to 0.

x = random l,u *uBasic*

x = random(l,u) *Lua*

Returns a random value within a range of `l` to `u`.

m = get_mode *uBasic*

Returns the camera mode: 0 = recording, 1 = playback, 2 = video.

rec,vid,mode = get_mode() *Lua*

Returns the camera mode:
`rec`: `true` = recording, `false` = playback
`vid`: `true` = video, `false` = photo
`mode`: binary value representing the CHDK shooting mode number (section 5.6).

r = set_capture_mode(i) *Lua*

Sets the shooting mode. Parameter `i` specifies the CHDK shooting mode number. `true` is returned on success, `false` otherwise. See also section 5.5.12 for high-level access.

is_capture_mode_valid(i) *Lua*

Returns `true` if the parameter `i` is a valid CHDK shooting mode number (section 5.6)

m = get_drive_mode *uBasic*

m = get_drive_mode() *Lua*

Returns the camera's drive mode: 0 = single shot, 1 = series, > 1 = timer.

m = get_IS_mode *uBasic*

m = get_IS_mode() *Lua*

Returns the mode of the image stabilizer (IS): 0 = continuous, 1 = when shooting, 2 = panning, 3 = off.

m = get_movie_status *uBasic*

m = get_movie_status() *Lua*

Returns the current status of the video function: 0 = stop, 1 = pause, 4 = record, 5 = save.

set_movie_status m *uBasic*

set_movie_status(m) *Lua*

Sets the current status of the video function: 1 = pause, 2 = restart, 3 = stop.

s = get_shooting *uBasic*

s = get_shooting() *Lua*

Returns 1 resp. `true` if the camera is currently shooting or has prepared for shooting and is ready to shoot.

x = get_propset *uBasic*

x = get_propset() *Lua*

Returns the type of operating system: 1 = VxWorks, 2 = DryOS.

v = get_prop p *uBasic*

v = get_prop(p) *Lua*

set_prop p,v *uBasic*

set_prop(p,v) *Lua*

Retrieves and sets properties (section 5.6).

b = get_vbatt *uBasic*

b = get_vbatt() *Lua*

Returns the battery voltage into variable `b` (in milliVolts).

t = get_temperature p *uBasic*

t = get_temperature(p) *Lua*

Returns a temperature in degrees Celsius. The variable `p` specifies the probe: 0 = optics, 1 = CCD, 2 = battery.

u = get_usb_power *uBasic*

u = get_usb_power() *Lua*

Sets variable `u` to a value > 0 if the camera detects a signal on the USB V+ pin. The returned value is the duration of the signal in units of 10 milliseconds. Typically, this command is used to implement advanced remote control functions (section 5.7.5).

exit_alt *uBasic*

exit_alt() *Lua*

Leaves the CHDK-specific <*ALT*>-mode (chapter 4). Please note that leaving the <*ALT*>-mode interrupts script execution. The script execution is resumed when the camera returns to <*ALT*>-mode.

shut_down *uBasic*

shut_down() *Lua*

The camera is switched off as soon as possible.

5.5.11 Low-level commands (Lua only)

value = peek(address)

Fetches memory content from the specified address.

Note: *If the address is outside the address range of the camera, the operating system will crash!*

status = poke(address,value)

Writes the specified value into memory at the specified address.

Warning! *Make sure you know exactly what you are doing. Writing wrong memory content could crash the operating system or even damage your camera!*

num = get_flash_params_count()

Returns the number of parameters in flash memory. Canon cameras feature a nonvolatile flash memory to store data even when the battery is removed. The parameters in that memory are numbered starting from zero.

str,num = get_parameter_data(id)

Returns the value of a specified flash memory parameter. `id` identifies the parameter. The meaning of IDs may vary from platform to platform and is ongoing work. You can find a listing at http://chdk.wikia.com/wiki/Params.

`str` returns the parameter value as a string, which may contain nonprintable control characters.

`num` returns the parameter value as a number, provided the parameter length is not longer than 4 bytes. Otherwise `nil` is returned.

bitand(x,y), bitor(x,y), bitxor(x,y), bitshl(x,y), bitshri(x,y), bishru(x,y), bitnot(x)

These commands perform bitwise operations on integers: `AND`, `OR`, `XOR`, `SHIFT LEFT`, `SHIFT RIGHT`, `SHIFT RIGHT UNSIGNED`, `NEGATION`.

5.5.12 The library capmode.lua (*Lua* only)

The library `capmode.lua` is contained in the full CHDK distribution and simplifies access to Canon capture modes such as `AUTO`, `P`, `TV`, `AV`, `M`, `PORTRAIT`, `NIGHT`, `LANDSCAPE`, ..., `EASY`, `SCN_DIGITAL_MACRO`, and `SCN_STITCH`. To use these commands, you must load the library with the following instruction:

```
capmode = require("capmode")
```

capmode.mode_to_name

Lua table that maps CHDK shooting mode numbers to mode names.

capmode.name_to_mode

Lua table that maps mode names to CHDK shooting mode numbers.

index = capmode.get()

Returns the CHDK shooting mode number (section 5.6). Returns 0 if the camera is in *Replay mode*.

name = capmode.get_name()

Returns the current shooting mode name (section 5.6). Returns `PLAY` if the camera is in *Replay mode*, `UNKNOWN` if the shooting mode is not known.

id = capmode.get_canon()

Returns the current Canon shooting mode ID (section 5.6). The values are ANDed bitwise with `0xFFFF` so that negative values are returned as `65536-id`.

r = capmode.set(modeid)

Sets the shooting mode specified in `modeid`. This can be a CHDK shooting mode number or a mode name. Returns `true` on success.

index = capmode.valid(modeid)

Returns the CHDK shooting mode number derived from `modeid`. It can be a CHDK shooting mode number or a mode name. `nil` is returned if `modeid` is not a valid shooting mode.

5.6 Property Cases

Both *uBasic* and *Lua* scripts feature commands for accessing so-called *Property Cases*. These are pieces of data used by the camera's operating system. Each property case is identified by a property ID (a number > = 0).

uBasic: **get_prop id**
Lua: **get_prop(id)**

and

uBasic: **set_prop id value**
Lua: **set_prop(id, value)**

Unfortunately, the property IDs differ between the *VxWorks* and *DryOS* operating systems (section 2.2), as you can see in the table below. In consequence, a script accessing properties and developed for Digic II cameras will not run on most Digic III and all Digic IV cameras, and vice versa. This is okay if you write a script for your own camera, but if you plan to publish your script in the CHDK community you should identify which cameras are supported.

If you want to make your script as widely applicable as possible, you should make it independent from the actual operating system. Under *uBasic,* you can use the command `get_propset` (section 5.5.10) to find out the operating system where your script is running. For example:

```
x = get_propset
if x = 1 then
  s = get_prop 205
else
  s = get_prop 206
endif
```

retrieves the shooting state of the camera under both operating systems.

A more elegant solution to this problem exists for the most important property cases under *Lua*. Your CHDK installation should already contain a library `propcase.lua` in folder `/CHDK/LUALIB`. This library maps symbolic names to property IDs and thus abstracts from the underlying operating system. The above query can be reformulated as:

```
props = require "propcase"
s = get_prop(props.SHOOTING)
```

You will find that many property cases shown in the table below are, in fact, covered by commands listed in section 5.5, so it's rarely necessary to access properties directly. The query shown above can be expressed more simply as:

```
s = get_shooting   rem uBasic
s = get_shooting() -- Lua
```

Not all of the property IDs and their values have been decoded yet—this is an ongoing community effort. In addition, not all findings have been verified for all camera types. So, the following table must be treated with some caution.

Vx Works	DryOS	*Lua*	Description
0		SHOOTING_MODE	**Shooting mode dial position**
236			18 = video (11)
	49	SHOOTING_MODE	**Shooting mode dial position**
	50		We list here only the most common shooting modes. A complete list of shooting modes is found in library `modelist.lua` in folder `CHDK/LUALIB/`. Some of the Canon shooting mode numbers (in front of the equality sign) differ from camera to camera. The latest findings are listed on http://chdk.wikia.com/wiki/Mode_dial_propcase_values. The values in parenthesis are the CHDK shooting mode numbers. –32768 = AUTO (1) –32764 = P (2) –32765 = Tv (3) –32766 = Av (4) –32767 = M (5) –32755 = PORTRAIT (6) –32757 = NIGHT (7) –32756 = LANDSCAPE (8) 2597 = VIDEO_STD (640/30fps) (9) 2598 = VIDEO_SPEED (320/60fps) (10) 2599 = VIDEO_COMPACT (160/15fps) (11) –32246 = Stitch Assist –32248 = Digital Macro

Vx Works	DryOS	*Lua*	Description
	49	SHOOTING_MODE	**Shooting mode dial position**
	50		We list here only the most common shooting modes. A complete list of shooting modes is found in library `modelist.lua` in folder `CHDK/LUALIB/`. Some of the Canon shooting mode numbers (in front of the equality sign) differ from camera to camera. The latest findings are listed on http://chdk.wikia.com/wiki/Mode_dial_propcase_values. The values in parenthesis are the CHDK shooting mode numbers. –32768 = AUTO (1) –32764 = P (2) –32765 = Tv (3) –32766 = Av (4) –32767 = M (5) –32755 = PORTRAIT (6) –32757 = NIGHT (7) –32756 = LANDSCAPE (8) 2597 = VIDEO_STD (640/30fps) (9) 2598 = VIDEO_SPEED (320/60fps) (10) 2599 = VIDEO_COMPACT (160/15fps) (11) –32246 = Stitch Assist –32248 = Digital Macro
1			**Photo effect**
2	225		**Sharpness for custom MyColors setting**
3	55		**Saturation for custom MyColors setting**
4	59		**Contrast for custom MyColors setting**
	227		**Long time exposure** 1 in “Night snapshot” scene mode or when exposure time is set to > = 1s, 0 otherwise.
5	268		**White balance** 0 = Auto 1 = Daylight 2 = Cloudy 3 = Tungsten 4 = Fluorescent 5 = Fluorescent H 6 = Flash 7 = Custom
6	102	DRIVE_MODE	**Drive mode** 0 = Single 1 = Continuous >1 = Timer

Vx Works	DryOS	*Lua*	Description
8			**Hi-speed continuous mode** 1 = OFF 0 = ON
9	155	METERING_MODE	**Metering mode** 0 = Eval 1 = Spot 2 = Center
11			**Macro** 0 = Normal 1 = Macro 2 = Super Macro
	6		**Focus mode** 0 = Normal 1 = Macro 3 = Infinity 4 = Manual 5 = Super Macro
12	133	FOCUS_MODE	**Manual focus mode** 0 = OFF 1 = ON
	8		**AF frame** 0 = AiAF 1 = Center 2 = FaceDetect
13	12		**AF mode** 0 = Single 1 = Continuous
14	224		**Delay of self-timer** (msec)
15	121	FLASH_ADJUST_MODE	**Flash adjust mode** 0 = Auto 1 = Manual
16	143	FLASH_MODE	**Flash mode** 0 = Auto 1 = ON 2 = OFF
	111		**External flash state** 0 = Absent 1 = Present and turned on 2 = Present and turned off
18	213		**Red-eye mode** 0 = OFF 1 = ON

Vx Works	DryOS	*Lua*	Description
19			**Flash slow sync** **0 = OFF** **1 = ON**
20	64	FLASH_SYNC_CURTAIN	**Flash sync curtain** 0 = First 1 = Second
21	149	ISO_MODE	**ISO value** 0 = ISO-AUTO 1 = ISO-HI > 1 = actual ISO
23	57	QUALITY	**Image quality** 0 = Superfine 1 = Fine 2 = Normal
24	218	RESOLUTION	**Image resolution** 0 = L 1 = M1 2 = M2 3 = M3 4 = S 5 = RAW (on G9) 6 = Postcard 8 = W
25 26	107 207	EV_CORRECTION_1 EV_CORRECTION_2	**EV correction** 96 units per stop. When setting EV correction, both properties must be set. A value larger than ±2 f-stops (192 units) is possible.
28	127		**Flash correction** 96 units per f-stop, if **Flash adjust mode** = Auto.
29	141	FLASH_MANUAL_OUTPUT	**Manual flash output** 0 = Low 1 = Medium 2 = Full if *Flash adjust mode = Manual*
32	4		**Exposure bracket range** 96 units per stop. A value larger than ±2 f-stops (192 units) is possible.
34	113		**Focus bracket range** 2 = Smallest 1 = Medium 0 = Largest
36	29	BRACKET_MODE	**Bracket mode** 0 = NONE 1 = Exposure 2 = Focus

Vx Works	DryOS	*Lua*	Description
37	219	ORIENTATION_SENSOR	**Orientation sensor** 0 = Normal 270 = Left 90 = Right
	21		**Auto rotate** 0 = OFF 1 = ON
	220		**Safety FE** 0 = OFF 1 = ON
	277		**Safety MF** 0 = OFF 1 = ON
39	26	USER_AV	**User-selected Av** 96 units per f-stop, 0 = f/1, positive = smaller, negative = wider.
40	264	USER_TV	**User-selected Tv** 96 units per f-stop, 0 = 1 sec, positive = shorter, negative = longer.
52	233		**Stitch mode** 0 = Left to right 1 = Right to left 2 = Bottom to top 3 = Top to bottom 4 = Top left > top right > bottom left > bottom right
57	249	DIGITAL_ZOOM_POSITION	**Digital zoom steps** 0 = NONE >0 = number of steps. Digital zoom steps are camera-dependent.
58	94	DIGITAL_ZOOM_STATE	**Digital zoom state** 0 = OFF 1 = ON/default Other values are camera-dependent.
	95	DIGITAL_ZOOM_POSITION	**Digital zoom position** In digital zoom steps
63	5		**AF light** 1 = ON 2 = OFF
65	245 252	SUBJECT_DIST1	**Focus distance** In millimeters
66	65	SUBJECT_DIST2	Same
	254		–1 indicates infinity

Vx Works	DryOS	*Lua*	Description
67	115		**Focus OK, ready to shoot** 1 = Yes 0 = No
	18		**Number of green AF boxes after half-press** 0 indicates a focusing problem.
68	23	AV	**Effective Av** 96 units per f-stop
69	262	TV	**Effective Tv** 96 units per f-stop
70	79	DELTA_SV	
71	34	BV	**Scene brightness value (Bv)** 96 units per f-stop
72	246	SV_MARKET	**User-selected Sv** 96 units per f-stop
73	247	SV	**Effective Sv** 96 units per f-stop
74			**AE lock** 1 = ON 0 = OFF
76	103	OVEREXPOSURE	
77	24 25	MIN_AV	**Minimal available Av (max. Aperture)**
78	122 ???		**Flash fired** 0 = Not fired 1 = Fired
79	122 ???	FLASH_FIRE	**Fire flash** 0 = do not fire 1= fire
99	195	OPTICAL_ZOOM_POSITION	**Zoom step** Camera-dependent
100	269	WB_ADJ	**Color temperature** 1 unit = 6 °K
126	166		**Video frames per second (FPS)**
127	169		**Video x resolution** 0 = 160 1 = 320 2 = 640

Vx Works	DryOS	*Lua*	Description
128			**Video y resolution** 0 = 120 1 = 240 2 = 480
	170		**Video play mode** 0 = LP 1 = SP
	165		**Time-lapse movie shoot interval** In milliseconds
177			**Intervalometer** 0 = Intervalometer not active >0 = Number of current shots in sequence
178	117		**File numbering** 0 = Continuous 1 = Auto Reset
181	105	DISPLAY_MODE	**Display mode in record mode** 0 = Show info 1 = Do not show info 2 = LCD off 3 = EVF
	212		**Review info** 0 = Off 2 = Detailed 3 = Focus check
184			**Slideshow mode** 0 = Do not repeat 1 = Repeat
185			**Slide duration** 1 = 3s, 2 = 4s, 3 = 5s, 4 = 6s, 5 = 7s, 6 = 8s, 7 = 9s, 8 = 10s, 9 = 15s, 10 = 30s
186			**Print settings/DPOF mode** 1 = Standard 2 = Overview 3 = Both
187			**Print settings/DPOF date** 0 = No date 1 = Date
188			**Print settings/DPOF filenumber** 0 = No filenumber 1 = Filenumber
190	66		**Postcard mode date printing** 0 = No date 1 = Date 2 = Date & time

Vx Works	DryOS	*Lua*	Description
192			**AF frame/flexizone x position**
193	3		**Autofocus lock** 0 = Not active 1 = Active
194	92		**Digital zoom area** Used part of sensor with digital zoom. Can be used to determine digital zoom ratio.
196	61		**Language setting (L) and video output mode (V)** `256*L + V` **V:** 1 = NTSC 2 = PAL **L:** *(VxWorks/DryOS)* 0/0 = English 1/1 = German 2/2 = French 3/3 = Dutch 4/4 = Danish 5/5 = Finnish 6/6 = Italian 7/7 = Norwegian –/8 = Ukrainian 8/9 = Swedish 9/10 = Spanish 10/11 = Simplified Chinese 11/12 = Russian 12/13 = Portuguese 13/14 = Greek 14/15 = Polish 15/16 = Czech 16/17 = Hungarian 17/18 = Turkish 18/19 = Traditional Chinese 19/20 = Korean 20/21 = Thai 21/22 = Arabic –/23 = Romanian 22/24 = Japanese
200			**Selected movie mode**
205	206	`SHOOTING`	**Ready to shoot** Changes to 1 when camera is ready to shoot (focus and exposure set); changes back to 0 when shutter closes.

Vx Works	DryOS	*Lua*	Description
206	184		**"MyColors" mode** 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
207 208 209 210			**Red, green, blue, skin tone** for the "Custom" MyColors settting. A larger value means darker.
218			**Custom timer continuous** Number of shots to be taken
219	223		**Self-timer setting** 0 = 2 sec 1 = 10 sec 2 = Custom continuous
	63		**Number of continuous shots taken last time**
221	208	IS_FLASH_READY	**Flash ready** 0 = NO 1 = YES
223			**Microphone recording frequency** 0 = 11.025 kHz 1 = 22.050 kHz 2 = 44.100 kHz
227			**Microphone level** 1–5 = low - high
228			**Microphone wind protection** 0 = OFF 1 = ON
229	145	IS_MODE	**Image stabilizer (IS)** 0 = Continuous 1 = Shoot only 2 = Panning 3 = Off

Vx Works	DryOS	*Lua*	Description
230	60		**Converter** Indicates whether a wide angle or teleconverter is mounted. 0 = NONE 1 = Wide 2 = Tele
	91		**Digital teleconverter mode** Indicates whether a digital teleconverter is active. 0 = OFF >0 = ON
	280		**RAW+JPG** 0 = OFF 1 = ON
	290		**iContrast setting** 0 = OFF 1 = ON
	293		**Servo AF** 0 = OFF 1 = ON
	294	ASPECT_RATIO	**Widescreen** Indicates whether the camera is switched to the 16:9 aspect ratio. 0 = OFF 1 = ON
	296		**ND filter** Indicates whether the ND filter is active. 0 = OFF 1 = ON

Not every entry in the table is valid for all CHDK-supported cameras. You should check specific property cases for your camera before using them. You can do this with the help of the script `propdump.lua`, which you will find on the book CD. Execute the script once (section 5.1) to create a first *Property Dump*. Then modify the camera setting and execute the script a second time. By setting parameter `I`, you can compare the second property dump with the first one. The differences between them will be displayed on the screen and give you a hint as to which property relates to the changed camera setting.

PRESENTATION OF NEGATIVE VALUES IN *UBASIC*

When reading properties with the *uBasic* command `get_prop`, negative values are represented as so-called 2-complements. Depending on the storage size of the property (1 byte or 2 byte), either the value $2^{16}+x$ or $2^{8}+x$ is displayed.

Example: A value of 65500 is displayed. The real value is –36:

$2^{16}+(-36) = 65536-36 = 65500$.

In the table we have printed the property cases that can be *modified* by a script in **bold** letters. All other properties are read-only, or modifying them has not yet been tested. In general, however, it is better to change the camera settings by simulating key presses (section 5.1). Changing camera settings by setting property values can be sometimes problematic because only part of the camera may be affected by a specific property case. For instance, a property case may influence the camera optics while leaving the user interface untouched.

This example is a property-based version of the script `flashoff.lua` from section 5.1:

```
-- flashof2.lua by Berthold Daum
--[[
@title Flash off
]]
props = require "propcase"
set_prop(props.SHOOTING,2)
exit_alt()
```

Here, the flash is switched off by assigning a value of 2 to property 143 (property 16 on DIGIC II). The script works all right—the flash is indeed switched off. But the display does not reflect the new state. It still indicates that the flash is in *AUTO* mode. When using the camera in unattended mode (e.g., in an RC-controlled vehicle), nobody cares. But personally, I prefer a display that truly reflects the state of the camera, so I would choose the original version of the script as shown in section 5.1.

5.7 Example scripts

Scripting is a popular topic on the CHDK web pages. Most of the scripts deal with timers, bracketing, or motion detection. Timers are typically used for operating the camera autonomously (e.g., from a kite or a balloon) or

for creating time-lapse series, very often time-lapse movies. Bracketing scripts are used for creating *High Dynamic Range* (HDR) photos or for creating images with extended depth of field via focus stacks. Finally, motion detection is used for all kinds of purposes: lightning photography, surveillance, wildlife, and more. For example, I found it good fun to set up the camera at a party, load a motion detection script, and let the camera do the work. The resulting pictures are quite different from pictures taken manually: full of action and often hilarious.

In the following sections, we will present some scripts from these areas. All of them aim to be as independent of platform and camera model as possible, but you should first test them with your own camera to see if they are suitable. The main reason for presenting these scripts is so that you can learn something about scripting. After playing with the scripts found here or on http://chdk.wikia.com/wiki/UBASIC/Scripts, you may be able to create your own scripts and draw even more benefit from the CHDK.

5.7.1 Time machines

Why do you need a script shooting a series of photos? Doesn't your camera have a configurable series function? Of course it has.

But you probably have noticed that this function has its limitations. First, the number of shots is limited to 10. You may have an initial delay before the series begins but no delay between pictures, and exposure and focus are only measured once and not for each image. The latter can be a problem if the series function allows for long series with customizable delays. For example, during a long series, the illumination may change due to the sunset or a cloud hiding the sun.

There are quite a few reasons to look at time-lapse scripts—and in fact, most of the CHDK scripts available on the web are for time-lapse purposes.

We will present two scripts here. One simply takes a predefined number of shots with a predefined delay between each shot. It does this very precisely, even taking different shutter and image processing times into account. So, it will even keep to the schedule when exposure times become quite long. The other reads a predefined schedule from a text file and fires the camera at precisely the defined points in time. Such schedules are most useful for unmanned operations like balloon or kite photography, but they need to be prepared on a PC because the schedule must be supplied as a text file.

Both scripts are written in *Lua* because it is easier to get a precise timing in that language. A third script (also written in *Lua*) deals only indirectly with time series; it is used to rename images. That might be necessary for some time-lapse moviemakers. Here we also have the opportunity to try out *Lua* for file management.

Accurate time lapse

The *Accurate Time Lapse* script first defines a few parameters. `n` specifies the total number of shots to be taken. The delay between the shots is computed as `(((m*60)+s)*10+t)*100` from the parameters `m` (minutes), `s` (seconds), and `t` (tenth of a second). The result is the total delay in milliseconds.

Two more parameters allow control of focusing and display. If parameter `f` is set to 1, the camera will focus only once and then leave the focus unchanged. Otherwise, the autofocus process will be performed before each shot. If parameter `d` is set to 0, the display is darkened to save battery life.

```
--[[
@title Accurate time lapse
@param n Number of frames
@default n 10
@param m Delay min
@default m 5
@param s Delay sec
@default s 0
@param t Delay 1/10 sec
@default t 0
@param f Focus 0 Each 1 Start
@default f 0
@param d Display 0 off 1 on
@default d 1
--]]
```

In *Lua,* functions must be defined before they are invoked. So, we start with function definition first. Because there is no built-in command to set the display mode, we need to select it by simulated key presses. Since we cannot be sure about the initial state, we simply check after each key press to see whether we have reached the desired state. This is done with the function `get_prop()` that retrieves the respective property case (section 5.6). Instead of a numeric property ID, we use the constant `props.DISPLAY_MODE`. The table `props` is defined in library `propcase.lua`, which is part of the CHDK installation and which we have loaded with `require("propcase")`. The advantage of this technique is that it is platform-independent. The numeric Property Case IDs, in contrast, are different for each operating system. The `sleep()` function pauses the script for 100 msec to give the camera some time to do its work.

The definition of function `idiv()` and the `pcall()` expression provide compatibility with the PC-based debug environment (section 5.8). `idiv()` performs an integer division. Under the CHDK, the results of `idiv()` are identical to those obtained with the operator `/`; but in a floating-point

environment such as a PC, the operator / denotes a floating-point division. Because an integer division is what we want, and because it is more convenient to debug scripts on a PC, we use `idiv()` instead of / throughout the script.

```
function idiv(a,b)
  return (a-(a % b))/b
end
pcall(function()
    require("chdklib")
  end
)
props = require("propcase")

function set_display_mode(mode)
  while get_prop(
    props.DISPLAY_MODE) ~= mode do
    click("display")
    sleep(100)
  end
end
```

The implementation of the next function, `sleep_until()`, could be trivial: simply calling the `sleep()` command and specifying the interval between now and the target time. But here, we want to do a bit more and allow the user to abort the script with the **SET** key. By doing so, we have the ability to restore the focus and display settings, which we cannot do if the user aborts with the shutter button.

Therefore, we use the command `wait_click()` instead and specify the interval (`sleep_time`) as a timeout. When this timeout happens, function `is_pressed()` returns "no_key". We then return `false` because the user does not want to abort. If the user clicks the **SET** key, we return `true`. For any other key, we loop around and wait for the remaining time.

```
function sleep_until(ticks)
  repeat
    local sleep_time =
          ticks - get_tick_count()
    if sleep_time <= 0 then
      return false
    end
    wait_click(sleep_time)
    if is_pressed("set") then
      return true
    end
  until is_pressed("no_key")
  return false
end
```

The function focus() is called if the user wants to focus just once at the beginning of the series. In this case, the focusing is performed by half-pressing the shutter button. In the following repeat-loop we wait until the camera becomes ready to shoot. The sleep(1) command in that loop is recommended to give the camera processor some time to breathe. In *uBasic* that would not be necessary because the *uBasic* interpreter automatically waits 10 msec after each line of code. This is not the case for *Lua*—probably the main reason why *Lua* scripts are so much faster.

After focusing has finished, we lock the autofocus system and release the shutter button. The camera will not try to refocus until the lock is released.

```
function focus()
  press("shoot_half")
  repeat
    sleep(1)
  until get_shooting()
  set_aflock(1)
  release("shoot_half")
end
```

Function tohms() converts milliseconds into the more readable format h:m:s. It does so by dividing the seconds subsequently by 60 and keeping the remainders. Instead of using the operator / for division, we use the function idiv() defined above:

```
function tohms(ticks)
  local ts =
        idiv((ticks + 50), 100)
  local t = ts % 10
  ts = idiv(ts, 10)
  local s = ts % 60
  ts = idiv(ts, 60)
  local m = ts % 60
  local h = idiv(ts, 60)
  return h, m, s, t
end
```

All necessary functions are now implemented, and we can begin with the main program. First, we print a short note to the user describing how to abort the script. We then begin checking the parameters for valid values. If a value is invalid, the parameter is reset to its default value.

We can compute the interval between two shots (delay) as discussed above:

```
print "Press SET to exit"

-- Check parameters
if n <= 0 then n = 10 end
if m < 0 then m = 5 end
if s < 0 then s = 0 end
if t < 0 then t = 0 end
-- Compute delay value
delay = (((m*60)+s)*10+t)*100
```

If focusing is only desired at the beginning of the series, we invoke function `focus()` now. It performs the focusing and then locks the focusing system.

Then we save the current display mode in order to restore it at the end of the script. If a dark display is desired, we use function `set_display_mode(2)` to switch the display off.

```
if f > 0 then focus() end
old_display_mode =
  get_prop(props.DISPLAY_MODE)
if d <= 0 then
    set_display_mode(2)
end
```

Now we can initialize the main loop. We get the start time in milliseconds (since camera start) and initialize the variable `next_event` with that value.

Within the loop we shoot an image, then increment the variable `next_event` with the computed delay. If the display is active, we print a log entry to inform the user about the progress. To do so, we use the function `format()` from the *Lua* string library (section 5.4.13). Then we sleep until the time computed in `next_event`. If the user has clicked **FUNC/SET**, we break the loop and finish the script. This, of course, also happens when the frame counter has reached the predefined number of shots.

Finally, the display mode is reset to its original state and the AF lock is released, just in case it was locked by the `focus()` function.

```
start_ticks = get_tick_count()
next_event = start_ticks
for frame = 1, n, 1 do
  shoot()
  next_event = next_event + delay
  if d>0 then
    h, m, s, t =
    tohms(next_event-start_ticks)
    print(string.format(
```

```
        "%u of %u, MET %u:%u:%u.%u",
        frame, n, h, m, s, t))
    end
    if sleep_until(next_event) then
      print "Aborted"
      break
    end
  end
  -- restore display mode
  set_display_mode(old_display_mode)
  -- restore AF lock mode
  set_aflock(0)
```

This script is very precise because it always computes the time of the next event and determines the sleep time as the difference between the next event and the current time. Even long exposure times do not disturb the script's timing—provided, of course, the exposure times don't exceed the defined delays between shots.

Time-lapse movies

Time-lapse scripts are often used to create time-lapse movies. Of course, your camera probably offers a built-in function to create time-lapse movies, but it is rather limited. Let's see what's possible with a time-lapse script.

First, with a script you don't work in video mode but in photo mode. This allows you to step up the image resolution. *High Definition* (HD) movies (1600x1200 pixels) become possible, and wide-screen movies (16:9) are no problem. When creating a time-lapse movie in this way, you should shoot in JPEG mode (not RAW). RAW files take up too much space, and the programs used for composing the movie expect JPEG.

Select an image size large enough to cover the output format. Output for normal television (NTSC) needs a resolution of at least 720 x 480 pixels, so the `M3` image size (1600x1200) should be more than enough. The same image size is also sufficient for 4:3 *High Definition* (HD) movies. However, if you plan to create a wide-screen movie (16:9), you should use image size `W` instead. *Wide-screen HD* requires at least 1920x1080 pixels, so image size `W` is more than enough. In addition, your display is switched to the wide-screen format when using image size `W`.

Both formats (`M3` and `W`) offer you different compression ratios. I prefer to start at medium resolution. The compression ratio "Superfine" is hardly necessary for a movie. With a 4 GB card, you can expect the following play times at 30 fps:

	M3	W
Superfine	119 sec	48 sec
Medium	202 sec	78 sec
Low	406 sec	156 sec

When shooting at intervals of one minute, a 4 GB card can cover a minimum of one day and a maximum of 8 days 11 hours. An external power supply is highly recommended for such a task. Changing batteries during a shooting session can easily ruin the whole set-up.

By the way, long time-lapse sessions can wear out your camera. A movie with five minutes' playtime at 30 fps consists of 9,000 single shots. The fewer moving parts your camera has, the better. (Using a DSLR with its complex mechanics and mechanical aperture is not a good idea.) A small compact camera with an electronic shutter and a neutral density filter is definitely preferred.

Also, be careful about the *Digital Zoom*. When shooting HD movies, you should avoid the *Digital Zoom* altogether or only use it up to approximately a factor of 2. Otherwise, you can use any camera feature and any CHDK feature that you would use when shooting manually. You can allow the camera to determine the exposure and focus for each new shot, or you can use *Overrides* (section 4.3.1) to set fixed exposure values and/or a fixed subject distance. You can use extreme exposure times (sections 4.3.5 and 4.3.6). You can even use curves (section 4.3.8) to compress contrast or do some wild, experimental stuff. You can bracket every shot, later applying tone mapping to each exposure group to create an HDR movie. There are, in fact, some specialized scripts for that task in the community, and we will discuss this technique in section 6.3.

After a time-lapse session, you will end up with a series of photos—not with a video. Additional post-processing steps are required to convert the photos into a movie:

- First, some resampling might be required, depending on the selected image size during shooting and the target output format. You will need a tool that is able to downsize images in a batch process. There are a number of tools for this purpose. If you have *Photoshop,* you can use *Actions*. Similarly, in *Paint Shop Pro* you would use *Scripts,* and in *Picture Window* you would use *Workflow*. Free tools with batch facilities include *IrfanView, Picasa,* and the command-line tool *Image Magick*. On the Mac, you can create an *Automator* action.
- Finally, you must feed the images into a time-lapse converter. Suitable programs are *Quicktime Pro, iMovie* (Mac), *MovieSalsa, MakeAVI* (free),

PhotoLapse (free), *JPGVideo* (free), or the free command-line tool *Ffmpeg*. The last one is a cross-platform tool that requires the photos to be numbered sequentially starting at 1 (e.g., `frame_1.jpg`, `frame_2.jpg`, `frame_3.jpg`, ...). *IrfanView* allows you to do batch renaming of photos. Alternatively, you could easily rename the images within the camera before transferring them to the PC. The following *Lua* script is all you need.

Renaming files

The following script can be used for renaming files. It uses "FRAME_" as a filename prefix and starts numbering at 1. However, this can be easily changed.

```
--[[
@title Rename files
@param s start value
@default s 1
--]]
root = "A/DCIM"
prefix = "FRAME_"
suffix = ".JPG"
pcall(function()
      require("chdklib")
      root = "TEST"
    end
    )
```

The header defines the script title and a parameter for the start value of the sequence number. The directory root is set at `A/DCIM`, where the subfolders for the images are located. The following `pcall()` expression is used for testing the script on a PC (section 5.8), where we also use a different root directory.

The function `renameFile()` does the actual renaming with the help of the operating system function `os.rename()`. But before we do so, we check to see whether the file to be renamed is actually a file (and not a folder). We also check to see whether a file with the new name already exists. In that case, we throw an error message because giving a file the name of an already existing file can cause unexpected results. Function `os.stat()` is used to do all that. It returns status information about the specified file (section 5.4.13).

In case of success, we increment the image counter s.

```
function renameFile(path, file)
  oldname = path..file
  local t = os.stat(oldname)
  if t["is_file"] then
    newname =
         path..prefix..s..suffix
    t = os.stat(newname)
    if t then
      error(
       oldname.." already exists")
    end
    r, msg = os.rename(
           oldname, newname)
    if not r then
      error(msg)
    end
    s = s + 1
  end
end
```

The following code is the main script part. It first lets the user select a subfolder of DCIM/. It does so by fetching the members of the root directory and prompting the user with the subfolder names. The user can navigate with the **LEFT** and **RIGHT** buttons to the subfolder whose files shall be renamed. The user selects the subfolder by pressing **FUNC/SET**. The button **DISP** is used to abort.

```
-- folder selection
  files, msg =
            os.listdir(root,false)
  if not files then
    error(msg)
  elseif not files[1] then
    error("No image folders")
  end
  i = 1
  while true do
    cls()
    print("Press SET to select")
    print("LEFT, RIGHT to scroll")
    print("DISP to abort")
    print("Folder: "..files[i])
```

```
    wait_click(5000)
    if is_pressed("set") then
      break
    elseif
        is_pressed("display") then
      i = nil
      break
    elseif is_pressed("left") then
      i = i - 1
      if i <= 0 then
        i = #files
      end
    elseif is_pressed("right") then
      i = i + 1
      if i > #files then
        i = 1
      end
    end
  end
```

If the user selected a subfolder, the members of the subfolder are fetched and the `renameFile()` function is executed on each member.

```
  if i then
    path = root.."/"..files[i]
    files, msg =
           os.listdir(path, false)
    if not files then
      error(msg)
    elseif not files[1] then
      error("Folder is empty")
    end
    path = path.."/"
    for _, file in ipairs(files) do
      renameFile(path, file)
    end
  end
```

Of course, you could easily modify this script to implement your own numbering scheme. **Before trying out this (or your own) script, you should definitely make a backup copy of your memory card.**

Scheduled operation

The following script implements a quite different time machine. Instead of accepting a few parameters and taking photos in equal intervals, it allows for far more complex actions; it reads a schedule from a predefined text file and processes that schedule.

This has advantages: you can describe a far more complex time series than what is possible with a few parameters. There are disadvantages, too: before using the script, you must create a schedule file on the PC. Later changes in the field are not possible. To provide for some flexibility, I implemented the option to select among different schedule files. All schedule files are expected to be in folder `CHDK/SCHED/`. When you start the script, you can scroll through the content of that folder and select the schedule you wish to execute.

Each schedule file is created as a simple text file. Each line in the text describes a different time series. After a time series has stopped, the next line is executed. When there are no more lines left, the script stops. This is the syntax of a single line:

```
[F/A/I/mm ]met[ n i1[-i2]][I/N/D]
```

Let's examine what that means. The first clause is optional (denoted by the brackets) and specifies the focusing mode. Cameras with a manual focus option must be set manually to manual focus to allow for the options of that clause. The following values are possible:

- `F` causes the camera to determine the subject distance immediately and then lock the focus, so the autofocus is switched off during the subsequent shots.
- `A` removes the focus lock and allows the camera to autofocus again with each new shot.
- `I` sets the focus to infinity.
- `mm` sets the camera to the specified subject distance (in mm).

Next comes `met`, the *Mission Elapsed Time*. This entry defines when the series begins, measured from the time when the script was started. It is given in the format `[[h:]m:]s`. Seconds must be defined, but minutes and hours are optional.

The following clause is optional. In the case of a single shot, it can be omitted. In the case of a time series, `n` defines the number of shots. `i1` specifies the interval between the shots. Again, intervals are specified in the `[[h:]m:]s` format. Optionally, a second interval `i2` can be specified after a dash. If so, we have a sliding interval starting at `i1` and gradually changing to `i2` with each shot. This allows for accelerating and decelerating the time series—an interesting option for time-lapse movies.

The last (and optional) clause controls the display:

- `I` switches to the info display.
- `N` switches to the normal display (no info).
- `D` switches the display off to save power.

Lines that start with two dashes (`--`) are considered as comments and are printed to the display. Here are a few examples for script files:

```
-- Example
F 0 D
1:0 10 15
A 5:0 10 15-5
```

This schedule file results in the following actions:

1. "Example" is printed to the display.
2. The first shot in the schedule is taken immediately. The camera is focused before the shot and the focus is locked. The display is switched dark.
3. After one minute, the camera takes 10 shots with an interval of 15 seconds between shots.
4. Finally, at MET 00:05:00, the camera switches back to autofocus and takes another 10 shots starting with an interval of 15 seconds and ending with an interval of 5 seconds.

Now let's analyze the script and see how it works. It uses some elements that we developed already in the previous two scripts, such as file browsing, display mode switching, and exact timing. What is new here is reading file content and parsing (interpreting) that content. Basically, this is a script that reads and executes a script (the schedule file).

The header defines the script title. The schedule files are expected in folder `A/CHDK/SCHED`. The function `idiv()` implements integer division and is used instead of the standard division in order to achieve compatibility with the PC version of *Lua*. The following `pcall()` expression is the connection to the debug environment on the PC (section 5.8).

```
--[[
@title Scheduled timeseries
--]]
schedules = "A/CHDK/SCHED"
function idiv(a,b)
  return (a-(a%b))/b
end
```

```
pcall(function()
    require("chdklib")
    schedules = "SCHED"
  end
  )
```

The functions `set_display_mode()` and `sleep_until()` are not really new. They are simply copied from the *Accurate Time Lapse* script at the beginning of section 5.7.1.

```
props = require "propcase"

function set_display_mode(mode)
  while get_prop(props.DISPLAY_MODE)
                          ~= mode do
    click("display")
    sleep(100)
  end
end

function sleep_until(time)
  repeat
    local sleep_time =
          time - get_tick_count()
    if sleep_time <= 0 then
      return false
    end
    wait_click(sleep_time)
      if is_pressed("set") then
        return true
      end
  until is_pressed("no_key")
end
```

Function `focus()`, however, does a bit more than its sibling in the *Accurate Time Lapse* script. It performs the required focusing operations. In case of immediate focusing (`F`), the shutter button is pressed halfway to perform the focusing. The `repeat` loop waits until this process has completed. Then the autofocus system is locked. A parameter value of `A` releases the lock again. Other values are numeric and specify the subject distance. This is set with `set_focus()` after locking the autofocus system.

```
function focus(f)
  if f == "F" then
    press("shoot_half")
    repeat until get_shooting()
    set_aflock(1)
    release("shoot_half")
  elseif f == "A" then
    set_aflock(0)
  else
    set_aflock(1)
    set_focus(f)
  end
end
```

You probably missed parameter value `I`, which stands for infinity. This is taken care of in function `parseFocus()`, which analyzes the focus subclause as specified in a schedule file. It converts the value `I` to `-1`, a value that stands for infinity. Of course, it would also be possible to write `-1` directly into the schedule file, but the letter `I` is easier to remember and to write.

There are two more functions for parsing subclauses: `parseTime()` and `parseDisp()`. The function `parseTime()` accepts `[[h:]m:]s` expressions and converts them into milliseconds. To make parsing easier, it first appends a single colon to the end of the subclause. `parseDisp()` accepts the valid display operators and converts them into the display states required by function `set_display_mode()`.

```
function parseFocus(token)
  t = string.upper(token)
  if "A" == t or "F" == t then
    return t
  elseif "I" == t then
    return -1
  end
  return tonumber(t)
end

function parseTime(token)
  local t, s, a = 0, token..":", 1
  for j = 1, 3 do
    local e = string.find(s, ":", a)
    if not e then break end
    t = t * 60
```

```
    if e > a then
      t = t + tonumber(
              string.sub(s, a, e - 1))
    end
    a = e + 1
  end
  return t * 1000
end

function parseDisp(token)
  t = string.upper(token)
  d = string.find("IND",t)
  if d then return d - 1 end
end
```

The function `parse()` parses a whole line. It first checks to see whether a line is a comment line. If so, it prints the comment and returns `nil`. Otherwise, it sets the default values for the resulting parameters and appends a white space character to the end of the line to make parsing easier. Then the function searches for white space. If a white space character is found, the text before the white space is analyzed with one of the functions defined above. In case of success, the read pointer (`a`) is incremented, so that parsing continues behind the white space character.

```
function parse(line)
  if string.find(line, "%-%-")
                        == 1 then
    print(string.sub(
                line, 3, #line))
    return nil
  end
  line = line.." "
  local n,focus,disp,met,i1,i2,a =
       1,nil,  nil, 0,  0, 0, 1
  local e = string.find(line, " ")
  if (e - a) <= 1 then
    focus = parseFocus(
          string.sub(line, a, e - 1))
    if focus then
      a = e + 1
      e = string.find(line, " ", a)
    end
  end
```

```
  met = parseTime(
          string.sub(line, a, e - 1))
  a = e + 1
  e = string.find(line, " ", a)
  if e then
    local t =
           string.sub(line, a, e - 1)
    disp = parseDisp(t)
    if not disp then
      n = tonumber(t)
    end
    if disp or (not n) then
      return focus,met,1,0,0,disp
    end
    a = e + 1
    e = string.find(line, " ", a)
    local i1i2 =
            string.sub(line, a, e - 1)
    local p = string.find(i1i2,"%-")
    if p then
      i1 = parseTime(
           string.sub(i1i2, 1, p - 1))
      i2 = parseTime(string.
              sub(i1i2, p + 1, #i1i2))
    else
      i1 = parseTime(i1i2)
      i2 = i1
    end
    a = e + 1
    e = string.find(line, " ", a)
  end
  if e then
    disp = parseDisp(
           string.sub(line, a, e - 1))
  end
  return focus, met, n, i1, i2, disp
end
```

Finally, the function `tohms()` is already known from the *Accurate Time Lapse* script:

```
function tohms(ticks)
  local ts = idiv((ticks + 50), 100)
  local t = ts % 10
  ts = idiv(ts, 10)
  local s = ts % 60
  ts = idiv(ts, 60)
  local m = ts % 60
  local h = idiv(ts, 60)
  return h, m, s, t
end
```

Now the real script execution starts. The schedule files in folder `SCHED/` are fetched. The program then runs through a loop to let the user select one of those files. The index of the selected file is stored in variable `i`. We have already seen similar code in the script for file renaming.

```
files, msg =

         os.listdir(schedules,false)
if not files then
  error(msg)
elseif not files[1] then
  error("No schedule file")
end
i = 1
while true do
  cls()
  print("Press SET to select")
  print("LEFT, RIGHT to scroll")
  print("DISP to abort")
  print("Schedule: "..files[i])
  wait_click(5000)
  if is_pressed("set") then
    break
  elseif is_pressed("display") then
    i = nil
    break
  elseif is_pressed("left") then
    i = i - 1
    if i <= 0 then i = #files end
```

```
  elseif is_pressed("right") then
      i = i + 1
      if i > #files then i = 1 end
    end
  end
```

If a file was selected, we can now read the file and parse each line. But before we do so, we save the current display mode in order to restore it when the script has finished. We also capture the current time in the variable `start_ticks`. This will be the reference point for the *Mission Elapsed Time*.

The file is opened in read-only mode. The following `for` loop steps through all the lines and invokes `parse()` on each of them. The return values are assigned to the variables `foc`, `met`, `n`, `i1`, `i2`, and `disp`. Depending on whether a variable was supplied with a value or not (`nil`), the corresponding steps are executed.

Before we take a picture with `shoot()`, we wait until the computed event time (the start time plus the MET read from the file). If the user has pressed the **DISP** key to cancel, the loop is exited. After a shot, a log entry is created and printed to the display.

The `sleep_until()`, `shoot()`, and printing functions are contained in an inner loop which handles the time series defined by the current schedule line. For a single shot, `n` is 1, so this loop does not repeat. Otherwise, a new event time is computed. If `n` is greater than 2, we have more than one interval and must consider the possibility of sliding intervals, which we compute from `i1` and `i2`.

Finally, we close the schedule file, restore the display state, and—just in case—release the autofocus lock:

```
  if i then
    old_display_mode =
          get_prop(props.DISPLAY_MODE)
    start_ticks = get_tick_count()
    frame = 1
    file = io.open (
      schedules.."/"..files[i], "r")
    for line in file:lines() do
      local foc, met, n, i1,
              i2, disp = parse(line)
      if foc then
        focus(foc)
      end
      if disp then
        set_display_mode(disp)
      end
```

```
    if met then
      local next_event =
                start_ticks + met
      for k=1, n, 1 do
        if sleep_until(next_event)
                              then
          print "Aborted"
          next_event = nil
          break
        end
        shoot()
        h, m, s, t = tohms(
             next_event-start_ticks)
        print(string.format(
        "Frame %u, MET %u:%u:%u.%u",
                 frame, h, m, s, t))
        frame = frame + 1
        if n > 2 then
          next_event = next_event +
          idiv((n-k-1)*i1 +(k-1)*i2,
               n-2)
        else
          next_event = next_event + i1
        end
      end
      if not next_event then
        break
      end
    end
  end
  io.close(file)
  -- restore display mode
  set_display_mode(old_display_mode)
  -- restore AF lock mode
  set_aflock(0)
end
```

5.7.2 Bracketing

We already discussed bracketing in section 4.6. While early CHDK versions relied mainly on scripts for bracketing, the newer builds of CHDK provide enough built-in support for most bracketing applications. We can therefore keep this section short.

Where scripts can optimize the bracketing process is in the area of *Focus Stacking*. The standard CHDK function for bracketing (in *Extra Photo Operations>Bracketing in Continuous Mode>Subj.Dist.Bracket Value (MF))* uses an increment (in mm) between the different shots of a series. This increment increases in proportion to the current focusing distance. If you specify 100 mm as a bracket value and start at a distance of 100 cm, the camera will shoot a series of photos with 100cm, 110cm, 121cm, 133.1cm, ... This is not bad, but it would be better to increment the focusing distance based on the actual *Depth of Field* (DOF). This range increases with the focal distance and not at all in linear proportions. In addition, the DOF depends on the aperture and the focal length. A script can utilize the DOF and produce an optimized series of photos.

The CHDK is able to compute the total DOF, the near limit, and the far limit. These values can be shown on the display (section 4.2.8), but they are also accessible in scripts. Unfortunately, we cannot use them in this script. When the camera is in normal focusing mode, values for the near and far limits that would fall into the macro range are returned as –1 (infinity). And vice versa—when the camera is in macro mode, values outside the macro range are returned as –1. However, we want a script that covers the full range of distances from close-up to infinity. Fortunately, it is quite easy to compute the far limit for each subject distance. Given the subject distance `f` and the hyperfocal distance `y`, the far limit `g` is computed as:

```
g = y*f / (y-f)
```

provided that `f` is smaller than `y`. The hyperfocal distance is easily obtained from the CHDK with:

```
y = get_hyp_dist
```

When starting at close range, we can therefore compute the far limit `g` for each shot and use this value as the new subject distance for the next shot. We stop as soon as we reach a specified end value or infinity.

Working this way, we get a series of shots with nicely overlapping sharpness ranges while keeping the number of images at a minimum.

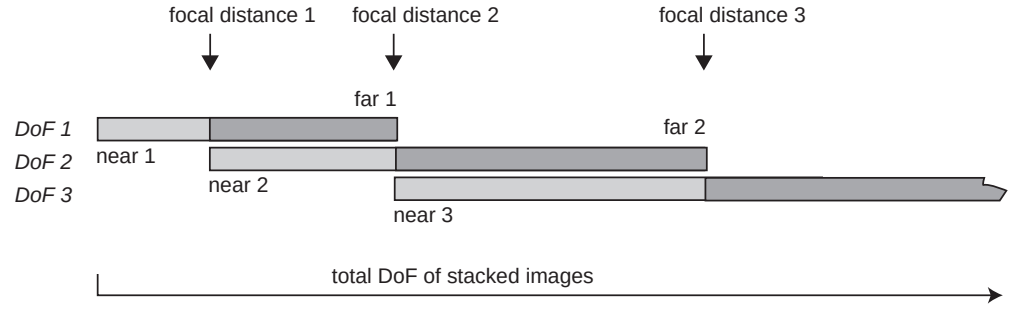


Figure 5-3
Adaptive focus stacking with the distance settings depending on the DOF.

The following script accepts four parameters: two for the near limit and two for the far limit of the composite sharpness range. Each limit is described by a base value (in millimeters) and a factor (1, 10, 100). You don't have to click 1,000 times for a distance of one meter. Instead, you dial a base value of 10 and a factor of 100, which is done with just a few clicks. The far limit factor, in addition, allows dialing in *infinity* directly. The default range is set to (20 cm - infinity). Because there are no timing constraints for this script, we have used *uBasic* to implement it.

```
@title Adaptive Focus Stacking
@param a close distance (mm)
@default a 2
@param b factor (0=1,1=10,2=100)
@default b 2
@param c far distance (mm)
@default c 5
@param d factor (0=1,1=10,2=100,3=inf)
@default d 3
```

In the following instructions, we make sure that no parameters are invalid. Invalid parameters (negative or zero distance) are set to their default values.

```
rem consolidate parameters
if a < 1 then a = 2
if c < 1 then c = 5
```

Then the script computes the true limits in millimeters by multiplying the base values with their factors. Only in the case of infinity do we directly assign the value of –1 that represents infinity.

```
rem compute distances in mm
select b
case 0;
case 1;a = a * 10
case_else a = a * 100
end_select
select d
case 0;
case 1;c = c * 10
case 2;c = c * 100
case_else c = -1 rem infinite
end_select
```

The next instructions are used to turn the automatic flash off. Flash illumination is not really desirable for DOF stacking. In addition, the flash operation can make the script behave unpredictably because of the time needed by the camera to recharge the flash. So, we store the current flash mode into variable `m` and then call subroutine `fmode`. The desired flash mode (2 for off) is set to transfer variable `M`.

```
rem turn off flash
m = get_flash_mode
M = 2
gosub "fmode"
```

Then, the shutter button is half-pressed. This is done to determine the correct exposure. If the camera is in autofocus mode, the focus distance is also determined. The subroutine `prep` waits until the camera is ready to shoot. The focus distance is retrieved and kept in variable `z` so that we can later restore the focus. We also retrieve the hyperfocal distance and keep it in variable `y`.

As a final preparation step, we lock the autofocus. During the following shots, the autofocus facility will not be used. This includes the AF light, too, which will not be illuminated.

```
press "shoot_half"
rem focus and exposure
gosub "prep"
z = get_focus
y = get_hyp_dist
print "Prior distance:",z
set_aflock 1
```

Now we can start shooting. The current focal distance is kept in variable `f` which is set to the near limit (parameter `a`). The shooting distance is printed on the screen to inform the user about the shooting progress. The focus is set and the script is paused for a while to give the camera some time to adjust the optics. The amount of time needed may depend on the camera model. After this pause, the camera is fired.

Then we wait until the camera has processed the image (subroutine `wait`). If the subject distance `f` was set to infinity or is larger than the hyperfocal distance, we are done; a longer focus is not possible or does not result in better sharpness. Otherwise, the DOF far limit of the current shot is determined with the above formula. This value is used as the new focal length:

```
rem DOF series
f = a
n = 0
do
   if f < 0 then
     print "dist: inf"
   else
     print "dist:",f
   endif
   set_focus f
   sleep 3000
   click "shoot_full"
   n = n + 1
   gosub "wait"
   if f < 0 or f >= y then goto "exit"
   f = y*f / (y-f)
   if f >= 65535 then f = -1
until f > c and c >= 0
```

After completing all shots, the camera is reset to its previous state. The shutter button (which was still half-pressed) and the AF lock are released, the camera is reset to its previous focal distance, and the flash is switched back to its previous mode. Finally, we print a short summary and play a timer sound to wake up the user. The `wait_click` allows the user 30 seconds to read the screen:

```
:exit
set_aflock 0
release "shoot_half"
set_focus z
M = m
```

```
gosub "fmode"
print "done:",n,"shots"
playsound 3 rem timer sound
wait_click 30000
end
```

The subroutine `fmode` checks the current flash mode. If it is not the desired flash mode (passed in transfer variable `M`), it fires the necessary key presses to advance to the next item in the flash menu (section 5.1):

```
:fmode
r = get_flash_mode
while r <> M
      click "right"
      click "right"
      click "set"
   r = get_flash_mode
wend
return
```

The subroutine `prep` is used to wait until the camera is ready to shoot—i.e., until the command `get_shooting` returns `1`. The subroutine `wait` works the other way around; it waits until the camera has processed the current shot:

```
:prep
   do
       r = get_shooting
   until r = 1
return

:wait
   do
       r = get_shooting
   until r = 0
return
```

A few words are necessary about script usage. First, cameras that support manual focusing must be switched to manual focusing mode. For other cameras, no action is required.

If your camera has a diaphragm, you should use an aperture two f-stops behind the open aperture. This will increase the DOF for each shot and thus reduce the number of shots that need to be taken. When shooting with a fully open aperture (and this is always the case for cameras that have only an ND filter), a large number of images might be necessary to cover the

entire specified area of sharpness. This is particularly true for macro and telephoto work. Make sure you have enough space on your memory card and use fully loaded batteries.

Figure 5-4
Breuburg Castle, Germany. A picture like this cannot be obtained in a single shot with a camera that has an ND filter but no diaphragm. The *Adaptive Focus Stacking* script was used to shoot a series of differently focused images. The settings a=45, b=1, d=3 resulted in a focal range of 45 cm to infinity. The six shots were then composed in *CombineZP*. Some retouching was required for the clouds that were moving fast between the shots, resulting in a staggered effect.

5.7.3 Motion detection

Motion detection is one of the great features of the CHDK. This feature makes use of the camera's hardware motion detection facilities. The camera provides such facilities for its own purposes: to enable image stabilization and to adjust exposure time in regard to subject movements. The CHDK uses them to trigger events—in most cases, to fire a shot. But if you like, you can play a sound instead. CHDK motion detection has many useful applications, such as lightning photography, wildlife and sports, surveillance, and more.

Basics

Motion detection is enabled by the CHDK command `md_detect_motion`.

uBasic:

```
md_detect_motion a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p
```

Lua:

```
h = md_detect_motion(a, b, c, d, e, f, g, nil, i, j, k, l, m, n,
o, p)
```

The large number of parameters indicates that this command is highly configurable. Let's see what these parameters mean:

a,b	**Number of columns and rows** the picture is divided into. By dividing the picture into cells, motion can be detected in a particular image part (see variables i, j, k, l, m). A fine grid enables the camera to detect the motion of small objects, but the camera will also react more slowly to movements.
c	**Color component** observed. 0 = U channel (chroma, green-blue) in the YUV-color model 1 = Y channel (luma) in the YUV-color model 2 = V channel (chroma, red-green) in the YUV-color model 3 = R channel (red) in the RGB color model 4 = G channel (green) in the RGB color model 5 = B channel (blue) in the RGB color model
d	**Timeout** in milliseconds. After the specified time, the next *uBasic* command will be executed even if no motion is detected.
e	**Time interval** (msec) between frames compared. Increase this value to detect slow movements.
f	**Threshold value**. If a change in at least one cell is larger than this threshold value, an event is fired.
g	**Grid display** on the screen. 0 = no cell grid is drawn 1 = cell grid is drawn
h	**Return variable**. Contains the number of cells where motion was detected.
i	**Masking mode** 0 = nothing masked 1 = everything inside of (j,k,l,m) 2 = everything outside of (j,k,l,m)
j,k	**First column and row of mask**
l,m	**Last column and row of mask**
n	**Shutter mode** 0 = leave shooting to script 1 = immediate shoot without focusing (fastest shooting option) 9 = don't release shutter on immediate shoot (leave that to script) *Under Lua, n = 0 should always be used. Lua scripts are fast enough and don't require this work-around for the slow uBasic interpreter.*
o	**Sub-sampling**. Analyze only every o-th pixel. Higher values improve speed but reduce accuracy.
p	**Delay** in milliseconds before triggering starts. Should be zero for lightning photography. Otherwise, allow some delay for calibration.

An additional command allows for a finer analysis of the detected motion:

```
md_get_cell_diff i,j,k      Basic
k = md_get_cell_diff(i,j)   Lua
```

Here, the variable `k` will contain the change in the `i`-th cell and the `j`-th row. The value of `k` will range from 0 to 255. This command could be used, for example, after the command `md_detect_motion` in order to take a shot when the detected motion has stopped.

The script

The following script example, written in *Lua,* is a universal script for motion detection. Of course, on the Internet—and in particular, on the CHDK website—you will find many similar scripts, and I've drawn some inspiration from a number of them. All of these scripts are based on the powerful CHDK command `md_detect_motion`, which we have discussed above. This command alone has a dozen parameters. Add additional parameters for prefocusing and timing, and you end up with 15+ parameters.

This is anything but practical when working in the field—even when working with parameter sets as explained in section 5.1. In the field, it is better to have a selection of clearly named and predefined programs for different scene types, such as "Lightning", "Fast Motion", "Small Objects", "Macro", etc., that you simply select from a list.

This is exactly what the following script provides. The most commonly used scene programs are already predefined, but you can easily add additional scene programs by extending the script. When running the script, simply select the required scene program using the **RIGHT**, **LEFT,** and **FUNC/SET** keys. By selecting the scene program named "Parameters", you will still be able to control the script via individual parameters.

Now let's see how this script is constructed. First, the parameters are defined in the usual way. We have chosen the parameter names so that they match the names used in the description of the command `md_detect_motion` (see previous page). So, we only have to explain the additional parameters.

s	This parameter can be used in combination with any scene program. The asterisk in front of the parameter description indicates this. When set to *0,* the script will run in test mode—meaning that no pictures are taken. The value of 1, in contrast, means full action.
v	This parameter can also be used in combination with any scene program (indicated by the asterisk in front of the parameter description). The parameter is used for specifying a pause after each shot (in seconds). The value –1 has a special meaning: the script pauses until the user presses the **DISP** button.

t	Specifies the duration (in seconds) for video sequences or time series when the camera is switched to video or series mode.
u	This parameter is used for focus control: *–1* = Prefocus before motion is detected using autofocus (AF). *0* = Use autofocus immediately before each shot that is taken. *>0* = Prefocus with a fixed distance before motion is detected. u specifies the distance in centimeters. This can be used in combination with most scene programs. Please note that cameras with manual focus mode must first be switched to this mode when using focusing by distance.
z	This parameter is used for backlight control: 0 = The display backlight is completely switched off. 1 = Backlight is switched off but is illuminated again after a shot for image review. 2 = Backlight remains switched on.

We have also added an additional value option to parameter f (threshold or sensitivity). If f is negative, the threshold will be dynamically derived from the brightness value measured by the camera. Even then it is possible to influence how the threshold is derived from the brightness value: the more negative f is, the smaller the threshold will be, and the more sensitivity motion detection will register.

```
--[[
@title Motion detection
@param s *Test(0=test,1=real)
@default s 1
@param v *Pause(s)(-1=DISP)
@default v 0
rem Only for program >Parameters<
@param a # columns
@default a 4
@param b # rows
@default b 4
@param c Channel(0U,1Y,2V,3R,4G,5B)
@default c 1
@param d Timeout(s)
@default d 55
@param e Detection interval(ms)
@default e 1
@param f Threshold(0-255, <0=auto)
@default f -1
@param i Mask(0=no 1=in 2=ex)
@default i 1
@param j First col mask
```

```
@default j 2
@param k First row mask
@default k 2
@param l Last col mask
@default l 3
@param m Last row mask
@default m 3
@param o Subsampling (pixel)
@default o 6
@param p Delay (0.1s)
@default p 1
@param t Duration series/video
@default t 5
@param u *Focus(0=AF -1=PreAF >0=cm)
@default u -1
@param z Bcklght(0=off 1=revie 2=on)
@default z 2
]]
function idiv(a,b)
  return (a-(a%b))/b
end
pcall(function()
require("chdklib")
  end
)
```

The definition of the parameters in the script header section is followed by the definition of function `idiv()` and a `pcall()` expression that we already have seen in earlier scripts. These expressions allow testing and debugging the script on a PC (section 5.8). Next are the definitions of some basic functions needed by the main motion detection script:

```
function adapt(thresh)
  if thresh >= 0 then
    return thresh
  end
  local r = get_bv96()
  local w = 48 + thresh - idiv(r,20)
  if w < 12 then w = 12
  elseif w > 36 then w = 36
  end
  return w
end
```

The function adapt() is used to compute the threshold (sensitivity) depending on the scene brightness value measured by the camera. We get this value with the CHDK function get_bv96(). The following formula is heuristic and was found acceptable during trials. It may be changed if required. The resulting threshold value in variable w is then clipped between 12 and 36 and returned. If the incoming threshold has a positive value, it is simply returned—no dynamic shareholding is wanted in this case:

```
function focus()
  if u <= 0 or f < 0 then
    press("shoot_half")
  end
  if u > 0 then set_focus(u * 10) end
  repeat until get_shooting()
end
```

Function focus() is used for prefocusing or for immediate focusing. It works by pressing the shutter button halfway. This is also done in the case of dynamic shareholding (f<0) because the shutter button must be half-pressed to determine the scene brightness. If a fixed subject distance is specified (u>0), this distance is set. After all these preparations, the function waits until the camera is ready to shoot:

```
function series(duration)
  press("shoot_full")
  sleep(duration)
  release("shoot_full")
end

function video(duration)
  click("shoot_full")
  sleep(duration)
  click("shoot_full")
end
```

Triggering a time series or a video clip works a bit differently from taking a photo. For a time series, the shutter button must be pressed, then after a while released again. For a video clip, the shutter button must be clicked (pressed and released), and after a while it must be clicked again:

```
function pause(duration)
  if duration < 0 then
      repeat
             print("Continue: DISP")
             wait_click(3000)
      until is_pressed("display")
  else
      print(duration.."s pause")
      sleep(duration*1000)
  end
end

function wait()
   repeat until not get_shooting()
end

function wake()
  release("shoot_half")
  click("set")
  click("set")
end
```

The function `pause()` waits for the specified time. If a negative duration is specified, it waits for the **DISP** button to be clicked.

The function `wait()` simply waits until the camera has completed the processing of a shot. When this happens, the function `get_shooting()` returns false.

The function `wake()` is invoked in the case of a timeout when no motion is detected. It releases the shutter button so that it can be half-pressed again to update focus and exposure. Then it clicks the **FUNC/SET** button twice, which does nothing except prevent the camera from falling asleep.

Now we can start with the execution of the main script. The first step is to define the different scene programs. Here we use a powerful feature of *Lua:* the ability to store functions in tables. Each scene program is represented by a function. When you select a scene program, the corresponding function will be invoked and will set the script parameters as required. We will explain the predefined scene programs and their parameters at the end of the script.

There are two tables: the first contains the scene program names (labels); the second contains the corresponding functions. It is just a matter of adding additional labels and functions to extend this script with additional scene programs:

```
-- Program definition
labels = {"Parameters",
   "Lightning", "Fast",
   "Small Objects", "Macro",
   "Hat"
-- add additional labels here
}
programs = {
function()  --Parameters
end,
function() --Lightning
  a,b,c,d, e,f, i,o,p,t,z
= 4,4,1,55,1,10,0,6,0,2,1
  if u == 0 then u = -1 end
end,
function()  --Fast
  a,b,c,d, e,f, i,j,k,l,m,o,p,t,z
= 6,4,1,30,1,-5,1,2,2,5,3,6,1,5,2
  if u == 0 then u = -1 end
end,
function() --Small Objects
  a, b, c,d, e, f, i,j,k,l,m,o,p,t,z
= 12,12,1,55,50,-1,1,2,2,11,11,1,
  1,5,2
if u < 0 then u = 0 end
end,
function() -Macro
  a,b,c,d, e, f, i,j,k,l,m,o,p,t,z
= 5,5,3,55,10,-1,1,3,3,3,3,6,1,5,2
  if u == 0 then u = -1 end
end,
function() --Hat
  a,b,c,d, e, f, i,o, p,t,z
= 1,1,1,55,10,24,0,12,5,5,0
if u < 0 then u = 0 end
if v < 1 then v = 5 end
pause(v)
end
--[[add additional program implementations here]]
}
```

After we have populated the two tables with labels and functions, we can now allow the user to select one of them. The number of the scene program is stored in the variable `program`. The currently selected program label

is printed onto the display. The script then waits for a key press with `wait_click()`. Depending on the key clicked (**RIGHT** or **LEFT**), the value of `program` is decremented or incremented. When `program` becomes too small, it restarts at the largest program number (wraparound), and similarly, when it becomes too large, it restarts at 1. This whole section is executed in a loop that is only left when **FUNC/SET** is pressed. Even when `wait_click()` times out after 10 seconds, the script will loop around and wait for more key presses.

Finally, we invoke the function corresponding to the selected program. This function will assign new values to the script parameters and will thus configure the script for the selected scene program:

```
-- Program selection
program = 1
repeat
  cls()
  print("Program:LEFT/RIGHT/SET")
  print(">"..labels[program].."<")
  wait_click(10000)
  if is_pressed("left") then
    program = program-1
    if program < 1 then
      program =# programs
    end
  elseif is_pressed("right") then
    program = program + 1
    if program >=# programs then
      program = 1
    end
  end
until is_pressed("set")
-- Assign parms for selected program
programs[program]()
```

In the following section, we perform a sanity check for the parameters controlling the mask borders and the threshold. We simply check to see if the values are out of bounds and make sure that the low boundary is not higher than the high boundary. A threshold of 0 would be nonsense; the camera would fire without any motion:

```
-- Inhibit bad mask bounds
if m > b then m = b end
if m < 1 then m = 1 end
if k > m then k = m end
if k < 1 then k = 1 end
```

```
if l > a then l = a end
if l < 1 then l = 1 end
if j > l then j = l end
if j < 1 then j = 1 end
--Inhibit zero threshold
if f == 0 then f = -1 end
```

We now retrieve the drive mode from the camera. This is done by using the commands `get_drive_mode()` to detect *Continuous Mode* (`1`) and `get_mode()` to detect *Video Mode*. After this piece of code is executed, we have the following values in variable `s`:

0 debug, *1* single shot, *2* series, *3* video

```
-- check for drive mode
if s ~= 0 then
  s = 1
  if get_drive_mode() == 1 then
    s = 2
  end
  local _, video = get_mode()
  if video then s = 3 end
end
```

Variable `grid` is set to 1 if a mask is used (`i>0`), indicating that the grid is to be shown on the display. Finally, the variables `p` and `d` are converted to milliseconds:

```
-- scale and adapt parameters
if i > 0 then grid = 1 else grid = 0 end
p = p * 100
d = d * 1000
```

Now we are ready to start the event loop. At the very beginning, we check to see whether there is still space left on the memory card. If not, we break the loop and stop. If so, we print a short message about how to stop the script. The variable `shots` is a counter for performed shots.

Each time the event loop cycles, the camera is newly set up. If prefocus or a fixed distance is selected, the subroutine `focus()` is called to perform the focusing. The same is true when dynamic thresholding is selected, because the half-press of the shutter button performed in subroutine `focus()` is needed to measure the brightness of the scene. The threshold is then computed in subroutine `adapt()`. In the case of immediate focus (`u==0`), we release the shutter button half-press to be able to use it again immediately before the shot.

The variable z controls the display backlight. It is possible to switch the backlight off while waiting for a movement, or completely until the script is interrupted. By saving battery power, the camera can be used for longer surveillance tasks.

Then motion detection starts. The result is written into variable cells that contain the number of grid cells where motion is detected. If motion detection times out, cells contains the value *0*. When a motion is detected, a shot can be taken. If immediate autofocus (u==0) is selected, the subroutine focus is called again. Then the shot is performed by using the command press("shoot_full"). Note that the command press() and not the command click() is used so that the shutter button stays pressed. In the case of series and video sequences, the subroutines series and video are called instead. Finally, a message is printed to the screen and the image counter is incremented.

The backlight is switched on again, if desired. The script then pauses optionally and releases the shutter button. During that pause, the camera remains in review mode showing the image taken on the display. Then we wait until the camera is ready to shoot again. In the case of a timeout (cells==0), we perform the subroutine wake to stop the camera from powering down. The camera is now ready for the next cycle:

```
-- event loop
shots = 0
print("Stop: Shutter Button")
while true do
  if get_raw() then
    if get_raw_count() == 0 then
      break
    end
  else
    if get_jpg_count() == 0 then
      break
    end
  end
  if (s > 0 and s < 3 and u ~= 0)
                  or f < 0 then
    focus()
    if u == 0 then
      release("shoot_half")
    end
  end
  if z <= 1 then
    set_backlight(0)
  end
```

```
    local threshold = adapt(f)
    local cells = md_detect_motion(a,
          b, c, d, e, threshold, grid,
          nil, i, j, k, l, m, 0, o, p)
    if cells > 0 then
      if s > 0 then
        if u == 0 then focus() end
        if s == 1 then
          press("shoot_full")
        elseif s ==2 then
          series(t * 1000)
        else
          video(t * 1000)
        end
      else
        playsound(1) -- shutter sound
      end
      shots = shots + 1
      print(shots..": "..cells.."
        cells, Threshold: "..threshold)
    end
    if z == 1 or v < 0 then
      set_backlight(1)
    end
    if v ~= 0 then pause(v) end
    if cells > 0 then
      release("shoot_full")
      wait()
      cells = 0
    else
      wake() -- inhibit power down
    end
  end
```

This concludes the implementation of the motion script. Now let's see how the scene programs differ from each other and how to use them. When using the script, you should set the camera's native time-out for power-down to at least one minute.

Lightning

Lightning photography seems to be a passion among some photographers. The problem, however, is that thunderstorms are quite unpredictable. The *National Weather Service* (http://www.weather.gov/) and other Internet services may help to locate thunderstorms.

The next problem—after you have located a thunderstorm—is to catch a lightning flash. Simply opening the shutter for a longer period of time is not really a good option; if there are other light sources in the scene, the resulting picture might be overexposed.

Fortunately, practically all lightning flashes are preceded by a smaller preflash. This preflash is sufficient to trigger a lightning sensor and a connected camera. If the camera is fast enough, it will capture the main flash. Such lightning sensors are not cheap. Fortunately, most CHDK users don't need one; they can use the camera's image sensor for the same purpose.

The lightning scene program of our script is configured for speed. We have chosen a fairly coarse 4x4 grid, a short detection interval (1 millisecond), and no delay before shooting. By specifying a subsampling value of 6, we analyze only every sixth pixel, again resulting in fast operation. Also, we prefocus before motion detection starts unless a fixed distance is set in parameter `u`. Autofocusing directly before the shot would definitely be too slow to catch any lightning flash. Every 55 seconds (`d=55`), the motion detection command times out. In this case, the program loops around, does another cycle of prefocusing, and then waits again for motion. The script reacts to changes in the YUV brightness channel (`c=1`) and uses a low threshold for high sensitivity.

How should you expose for lightning? Using automatic exposure is out of the question. Instead, you should use *Overrides* (section 4.3.1). As a starting point, set the aperture to f/5.6 and the sensor speed to ISO 100. For cameras without a diaphragm, dial in the ND filter (which gives you an equivalent of f/8.0 at wide-angle zoom setting) and select ISO 200. Choose an exposure time that results in the desired scene rendering. Then put the camera on a tripod, start the script, select the *Lightning* program, press the FUNC/SET button, and wait.

WARNING! Lightning photography is dangerous. Each year lightning kills numerous people. Please refrain from photographing lightning in the open. If you happen to be outside during a thunderstorm, do not seek protection under trees. Instead, find a ditch or a surface depression and duck down with your legs closed. A safe place is also inside a car (not a convertible). The safest way to photograph lightning is from inside a building.

Fast movements

The settings for fast movements are similar to those for lightning. We use a slightly finer grid (4x6) and discard the border grid cells, so movements at the very border of the image are ignored. After the motion is detected, we allow for a small delay of 100 milliseconds for camera calibration. The threshold is determined from the brightness value measured by the camera, but with a higher sensitivity than normal. Because conditions may change rapidly, we set the time-out to 30 seconds so that the camera can more frequently adapt to altered conditions.

Small objects

The next program is used for detecting rather small movements or movements of small objects. For example, it is sufficiently sensitive to detect the movement of a hand in group photos.

This is possible by using a very fine grid of 12x12 = 144 cells. Again we discard the cells at the borders of the image. Motion detection with such a high number of cells performs much more slowly. We therefore adapt the other parameters of the program to slow operation. By using a subsampling value of 1, we analyze each single pixel, resulting in slower operation but working much more accurately. The program will catch slower movements than the previous programs because of a detection interval of 50 milliseconds. Finally, we focus immediately before the shot (`u=0`) if a fixed distance was not set in parameter `u`. Focusing immediately before the shot would not be suitable for fast movements (because it takes too long), but it results in more precisely focused images.

Macro

The macro program has a 5x5 grid where only the center cell is used for detection; all other cells are discarded. Only those movements happening in the very center of the image will be detected. Movements in the outer areas—typically movements of branches and leaves—are ignored. For the same reason, we only observe changes in the red channel (`c=3`). Movements of dominantly green foliage will be ignored, while movements of small red, orange, or brown subjects will be detected.

You may want to use a fixed subject distance (`u=...`) for this program or switch the camera to *Macro Mode*.

Figure 5-5
Despite its tattered wings, this bee is still busy collecting honey from a borage blossom. Photographed with the Macro Motion Detection program. Canon Digital Elph SD1100 IS, 38 mm (35mm equiv.), f/2.8, 1/430 sec, ISO 73. Shot in DNG format, developed with RawTherapee, somewhat cropped.

Hat

The last scene program, `>Hat<`, is not actually used to capture motion. Instead, it's used to capture unaware people. The camera is placed somewhere, the script is started, and a hat is placed over the camera to hide it. When something interesting happens, we lift the hat and the camera fires.

For this program, a simple 1x1 grid with a rather high threshold is fully sufficient. Of course, focusing must be performed immediately before the shot. We have also specified a high delay between motion detection and the actual shot to allow for the hat to be removed completely. There is a pause (normally five seconds) after the shot to allow the hat to be replaced. The script begins with that pause, so after starting the script you have time to cover the camera. In most cases, the camera should be set to autofocus and automatic exposure.

Benchmark

The big question is, how fast is motion detection? This depends on several factors: the camera, the script language used, and how well the script is tuned. One keen CHDK user has provided the community with an aid for determining the reaction time of motion detection scripts and cameras. Just open the URL http://dataghost.com/chdk/md_meter.html in your web browser and maximize the browser so that it fills the whole screen. Click the *Execute* link. This will start a script that quickly fills a sequence of table cells with black. After a while, the cells are made visible again and the cycle is repeated.

Now start a motion detection script on your camera and point the camera to the screen (a tripod is recommended). If you use the script developed in this section, choose the *lightning* scene program. If properly configured, the script will react every time the table cells become visible again. By the time the shutter is released, a few table cells will already be painted black. The first table cell that is visible in the recorded image tells you the reaction time.

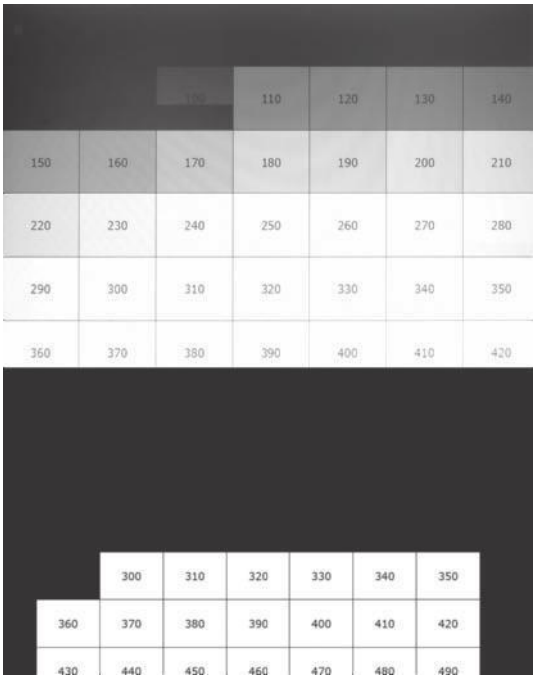


Figure 5-6
Top: Speed test with a Canon Digital Elph 1100 SD: the shutter starts to open after 100 ms. Bottom: The human user has no chance, even when using the computer's screen print facility. 300 ms was my best score. When shooting manual with the camera, we would have to add the camera's delay time on top of that (see below).

Here are the delays between motion and shutter release measured with an SD1100 IS and the scene programs implemented above. For comparison, we have added the reaction time when the camera is manually triggered.

Program	Delay	Remark
Lightning	100 msec	
Fast	150 msec	
Fine	1050 msec	Includes focusing
Macro	120 msec	
Hat	1000 msec	Includes focusing
Manual 1	400 msec	Shutter half-pressed
Manual 2	1200 msec	Includes focusing

5.7.4 Exposure control

The smaller Canon cameras are not equipped with diaphragms. The consequence is that the *Aperture value* (Av) cannot be changed. Therefore, these cameras do not have a shutter priority program where you preset a shutter time and the aperture is adapted.

The following script solves that problem in a different way. It modifies the speed of the sensor (aka the ISO value) to adapt the exposure to a preset shutter value. The shutter value can be preset through the CHDK *Overrides* (section 4.3.1). The script presented here switches the ND filter out. In bright conditions, you should use a rather fast shutter speed instead.

The script is written in *Lua*. The header defines a single parameter, `s`, used to specify the maximum acceptable ISO speed. If the scene is too dark for that ISO value and the preset shutter speed, the script will reduce the shutter speed rather than increasing the ISO speed above that limit.

As in previous examples, function `idiv()` and the `pcall()` expression are used to provide portability to a PC-based debug environment (section 5.8):

```
--[[
@title Shutter priority
@param s maxISO 1=50...8=6400
@default s 4
--]]
function idiv(a,b)
       return (a-(a % b))/b
end
pcall(function()
  require("chdklib")
  end
)
```

The variable `minIso` defines the *Sv96* value for the minimum acceptable ISO value (usually ISO 50). Should the scene be too bright for such a value, the script will increase the shutter speed rather than go below that value.

The following table, `corr`, contains correction values for shutter speeds of 1/1000 sec or shorter. Often, such short shutter speeds are not really precise. If desired, camera owners can calibrate their cameras by making test shots at 1/1000, 1/2000, 1/4000, and so on. The resulting values go into table `corr`. In function `compute_corr()`, the script reads values from that table and interpolates between those values for shutter speeds that do not exactly match the defined points:

```
minIso = 384 -- ISO 50
--[[ Correction values for
     high shutter speeds
     (unit = 1/96th f-stops).
  First value for 1/1000,
    second value for 1/2000, etc.
  + too dark, make brighter
  - too bright, make darker
]]

corr = {0,0,0,0,0,0,0,0,0,0}

function compute_corr(tv)
  if tv >= 960 then
    local i1 = idiv(tv-960, 96)
    local c1 = corr[i1]
    local c2 = corr[i1+1]
    return idiv(
           (c2-c1)*(tv%96),96) + c1
  else
    return 0
  end
end
```

All the prerequisites are now set up, and the script execution can begin. The parameter `s` is clipped to its limits, and the maximum acceptable *Sv96* is computed. Then the *Bv96* value is read by half-pressing the shutter button, waiting until the camera signals "ready to shoot", and reading out the *Bv96* value.

Next, the ND filter is switched out, the *Tv96* value is read, and the *Av96* value is read, too. This seems to be superfluous for cameras without diaphragms, but it isn't: the aperture changes with the focal length. At a telephoto setting, the aperture is usually smaller than at wide-angle setting:

```
if s < 1 then
 s = 1
elseif s > 8 then
 s = 8
end
maxSv = 288 + s * 96
press("shoot_half")
repeat
  sleep(1)
until get_shooting()
```

```
bv = get_bv96()
set_nd_filter(2)
tv = get_tv96()
av = get_av96()
```

Now, that we have *Bv96, Tv96,* and *Av96,* we can compute *Sv96* according to the APEX formula (section 4.2.7). There is one modification here: we allow for a user-defined correction value as discussed above. After *Sv96* is computed, the script checks to see whether it exceeds the maximum acceptable ISO value or is below the minimum possible ISO value. If yes, a recalculation is performed. Finally, the *Sv96* value is set, the camera is fired, and when the processing is done, the shutter button half-press is released:

```
sv = av + tv –
     bv + compute_corr(tv)
if sv > maxSv then
  tv = tv - (sv - maxSv)
  sv = av + tv –
       bv + compute_corr(tv)
end
if sv < minIso then
 tv = tv - (sv - minIso)
 sv = av + tv –
      bv + compute_corr(tv)
end

set_sv96(sv)
sleep(10)
click("shoot_full")
repeat
  sleep(1)
until not get_shooting()
release("shoot_half")
```

Before running this script, make sure that the values for *ND filter* and *ISO speed* are not overridden in the CHDK *Overrides*.

5.7.5 Remote control

While CHDK features basic functions for controlling the camera through the USB port (section 4.9), more sophisticated functionality can be achieved by combining USB control with a script. Typically, such a script would use the command `get_usb_power` to wait for an event from the remote control.

The signal can further be analyzed; the command returns the pulse length in units of 10 msec. So, different functions can be encoded using different pulse lengths.

The following script uses that functionality to allow zooming and shooting with a simple remote control as shown in section 4.9. A short click on the button of the remote control will fire the camera. For zooming, there are two modes that can be selected via parameter `m`:

- In *Step mode,* the remote button must be pressed longer than half a second to advance the zoom to the next position. The number of positions can be set via parameter `s`. (For `s <= 1`, zooming is disabled and any signal will fire the camera.) When the longest focal length is reached, the zoom direction will reverse until the shortest focal length is reached.
- In *Seconds mode,* the remote button must be pressed longer than half a second to zoom. The duration of the key press relates directly to the focal length: a longer press results in a longer focal length. The longest duration (for the longest focal length) can be set via parameter `s`. (For `s <= 0`, zooming is disabled and any signal will fire the camera.)

Optionally, the camera's backlight can be switched off while the script is running (parameter `b`). Another option (parameter `t`) is to avoid the camera powering down because of a time-out.

The script should run fine on most cameras since it automatically adapts to the different number of zoom steps as used by different cameras:

```
@title USB Remote Control
@param m Zoom:0=steps,1=duration
@default m 0
@param s Steps/seconds(>1)
@default s 4
@param b Backlight:0=off,1=on
@default b 1
@param t Timeout:0=no,1=yes
@default t 0
```

The number of zoom steps is first stored in variable `s`. This number varies for the different camera models (A-series: 9 or 15 zoom steps; S-series: 129 steps). When running in *Step mode,* the script first determines the current zoom step (`n`) and the best zoom direction (`d`). The zoom speed is set to prepare the camera for the `set_zoom` commands to come (section 5.5.4). Finally, variable `e` is initialized for time-out control:

```
if b > 0 then b = 1 else b = 0
z = get_zoom_steps
if m = 0 then
  s = s - 1 rem 0..s-1
  if z < s then
    s = z
  endif
  c = get_zoom
  n = (c * s + z/2) / z
  d = 1 rem zoom in
  if n >= s then
    d = -1 rem zoom out
  endif
endif
set_zoom_speed 100
e = get_day_seconds + 50
```

The main `while` loop contains an inner `do` loop that waits on a signal from the USB port. Within this loop, the display backlight is set to the desired state. If power-down is to be inhibited, the subroutine `wakeup` is called.

When a USB signal arrives (indicated by `p>0`), the duration of the pulse is analyzed. If the duration is shorter than 500 msec or if zooming had been disabled by setting parameter `s` to a value <= 1, the camera is fired. Otherwise, the subroutine `zoom` is executed. Afterwards, the script loops and waits for the next signal:

```
while 1
  do
    set_backlight b
    if t <= 0 then gosub "wakeup"
    p = get_usb_power
  until p > 0
  print "USB Pulse",p * 10;"msec"
  if p > 50 and s > 0 then
    gosub "zoom"
  else
    shoot
  endif
wend
end
```

The subroutine `wakeup` "tickles" the camera every 50 sec by pressing the **FUNC/SET** button twice. This will keep the camera from powering down. As

a matter of fact, it is essential that the camera's native time-out interval for power-down is set to a value of one minute or more.

```
:wakeup
  r = get_day_seconds
  if r > e then
    click "set"
    click "set"
    e = r + 50
  endif
return
```

Finally, the subroutine `zoom` is responsible for performing the zoom operation, depending on the mode parameter `m`:

- In *Step mode,* the current step value `n` is incremented by the value of the direction indicator `d`. If `n` hits the lower or upper limit, `d` is inverted. Then the physical step value is computed.
- In *Seconds mode,* the physical step value is computed directly from the pulse length.

After setting the new zoom level, a `sleep` command is issued to allow the camera to adjust the lens before new USB signals are accepted:

```
:zoom
  if m = 0 then
    rem Step mode
    n = n + d
    if n = 0 or n = s then
      d = -d
    endif
    c =  (n * z + s / 2) / s
  else
    rem Seconds mode
    i = s * 100 - 50
    c = ((p - 50) * z + i / 2) / i
    if c > z then
      c = z
    endif
  endif
  set_zoom c
  sleep 500
return
```

5.7.6 Configuration switching

The CHDK features an incredible number of different configuration parameters. Under shooting conditions, it is not always easy to keep an overview of which parameters are configured with which value. Wouldn't it be great if we could create some custom configurations—let's say for casual shooting, HDR work, panorama shooting, and kite aerial photography—and later simply switch between them? We have already seen something similar for the parameters of a script that we can save as a numbered parameter set and recall later (section 5.1).

The following script does exactly that. It uses the fact that all CHDK configuration parameters are stored in a single file, `CHDK/CCHDK.CFG`. When you invoke the script, it allows you to create a copy of this file, if the file has changed since the last copy was made. It also allows you to select one of the copies and make it the current configuration. After a restart of the camera, the selected configuration becomes the active configuration. Of course, every time the configuration is switched, a backup of the former configuration is made—just to be safe.

The script is written in *Lua* because it needs the file management and I/O functions of *Lua*. It maintains a small *INI* file (`CHDK/CONFSW.INI`) to remember the currently active configuration and a counter for the saved configurations. These are all stored in folder `CHDK/CONFIGS`. As in the previous examples, the `pcall()` expression provides compatibility with a PC-based debug environment (section 5.8).

```
--[[
@title Switch CHDK configuration
--]]
root = "A/CHDK"
pcall(function()
    require("chdklib")
    root = "TEST"
  end
  )

configs = root.."/CONFIGS"
ininame = root.."/CONFSW.INI"
current_config =
             root.."/CCHDK.CFG"
```

The INI file is created with the help of function `write_ini()`. This function accepts the parameters `name` with the name of the current configuration

and cnt with the current count of saved configurations. It writes two text lines that should look like this:

```
cnt = 1
cur = CONF1.CFG
```

To be safe, the new file is first written to file CONFSW.INI.NEW. If this is successful, the old *INI* file is renamed to CONFSW.INI.BAK (existing files of that name are deleted first), and file CONFSW.INI.NEW is renamed to CONFSW.INI. This is done in function _activate_file(). Working in this way, we always have a valid *INI* file. Even a battery going flat when writing the *INI* file cannot lead to a corrupt file:

```
function write_ini(name, cnt)
  local new = ininame..".NEW"
  local file,msg =
              io.open (new, "w")
  if not file then error(msg) end
  local ret,msg = file:write(
                 "cnt="..cnt.."\n")
  if ret then
    ret,msg = file:write (
                "cur="..name.."\n")
  end
  file:close()
  if ret then
    ret,msg =
        _activate_file(new, ininame)
  end
  if not ret then error(msg) end
end

function _activate_file(new, to)
  local bak = to..".BAK"
  os.remove(bak)
  os.rename(to,bak)
  local ret,msg =
                 os.rename(new,to)
  if not ret then
    os.rename(bak,to)
  end
  return ret,msg
end
```

Next is the definition of the table `parser_funcs` containing two functions. These functions will be used for parsing the entries of the *INI* file using regular expressions. The function with the key "count" checks to see whether a text line passed in the variable `line` has the syntax "cnt=..." where the characters following the equality sign must be decimal digits. At least one digit is required.

The function with the key "current" checks for text lines with the syntax "current=...". Here any characters are accepted after the equality sign. In both cases, the matching characters are captured (and therefore the parentheses) and assigned to the local variable `s`. In the case of success, the value of `s` is assigned to the variables `ccount` resp. `config_name` and the functions return `true`:

```
parser_funcs={
  count = function(line)
    local s = line:match(
                  "cnt = ([0-9]+)")
    if s then
      ccount = tonumber(s)
      return true
    end
  end,
  current = function(line)
    local s = line:match(
                   "cur=(.+)")
    if s then
      config_name = s
      return true
    end
  end,
}
```

The function `copy_file()` is used for copying configuration files. In this case, we first copy to an auxiliary file with the suffix ".NEW" and later rename it to the true target file name with the function `_activate_file()`. Again, we don't want to end up with a corrupted configuration file when power fails or the memory card is full.

The function first opens the source file in read mode and the target file in write mode. Because configuration files are not text files but binary files, we add the "b" option to the mode parameter of the `io.open()` function. The content of the source file is then read in chunks of 256 bytes and written to the target file. It is essential that we always close files that are successfully opened:

```
function copy_file(from, to)
  local new = to..".NEW"
  local ffile, msg =
              io.open (from, "rb")
  if not ffile then error(msg) end
  local tfile, msg =
              io.open (new, "wb")
  if not tfile then
    ffile:close()
    error(msg)
  end
  local ret = true
  while ret do
    local s = ffile:read (256)
    if not s then break end
    ret, msg = tfile:write(s)
  end
  tfile:close()
  ffile:close()
  if ret then
    ret,msg =
             _activate_file(new, to)
  end
  if not ret then error(msg) end
end
```

The next function, `compare_files()`, is used to compare the current configuration file with its former origin in folder `CONFIGS/`. The function returns true if the content of the file has changed since it was copied from its origin. If this is the case, we need to save the current configuration before we switch to a new configuration. The function compares both files by reading their contents in chunks of 256 bytes and comparing these strings:

```
function compare_files(n1, n2)
  local f1 = io.open (n1, "rb")
  if not f1 then
    return false
  end
  local f2 = io.open (n2, "rb")
  if not f2 then
    f1:close()
    return true
  end
```

```
  local ret = false
  while true do
    local s1 = f1:read (256)
    local s2 = f2:read (256)
    if not (s1 or s2) then
      break
    elseif s1 ~= s2 then
      ret = true
      break
    end
  end
  f1:close()
  f2:close()
  return ret
end
```

Now all preparations are done, and we can start to read the *INI* file. The configuration counter and the name of the current configuration are set to default values, and, if the *INI* file exists, it is read line-by-line using the `io.lines()` function. For each line, we run through the functions in table `parser_func` and check to see whether a function applies. By now we should have the current configuration count in the variable `ccount` and the current configuration name in the variable `config_name`:

```
ccount =  0
config_name = ""
-- ini file parsing
if os.stat(ininame) then
  for line in io.lines(ininame) do
    for _,f in pairs(
                    parser_funcs) do
      if f(line) then break end
    end
  end
end
```

Next, we allow the user to select the configuration to switch to. To do this, we fetch the list of files contained in the folder `CHDK/CONFIGS/`. If this folder does not exist, we just use the empty list.

If the list of files is not empty, we sort the files in alphabetic order to determine the index of the current configuration, since we want to display this configuration to the user. Then, in the following `while` loop, the user can use the **LEFT** and **RIGHT** buttons for scrolling in the list. The **SET** button can be used to select a configuration.

When showing a configuration to the user, we display not only its name, but also its modification date. This makes it easier for the user to identify the configuration:

```
-- config selection
files, msg = os.listdir(
                  configs,false)
if not files then files = {} end
if #files > 0 then
  table.sort(files)
  local i = 1
  while i <= #files do
    if files[i] == config_name then
      break
    end
    i = i + 1
  end
  while true do
    if i <= 0 then
      i = #files
    elseif i > #files then
      i = 1
    end
    cls()
    print("Press SET to select")
    print("LEFT/RIGHT to scroll")
    print("DISP to abort")
    local fstat = os.stat(
             configs.."/"..files[i])
    local mdate = fstat.mtime
    local s = os.date("%c", mdate)
    print("Switch to: ".."
             files[i].." ("..s..")")
    wait_click(5000)
    if is_pressed("set") then
      selected_config = files[i]
      break
    elseif
         is_pressed("display") then
      break
    elseif is_pressed("left") then
      i = i - 1
    elseif
         is_pressed("right") then
```

```
      i = i+1
    end
  end
else
  print(
        "No configs to choose from")
end
```

Before we switch to the selected configuration, we check to see whether the current configuration CHDK/CCHDK.CFG is different from the configuration remembered in the *INI* file. We do this by using the function compare_files() that we implemented above.

Then we ask the user whether to save the current configuration under a new name. If yes, we remember this decision in the variable newconf. If no *INI* file exists, we save the current configuration by default:

```
if #config_name > 0 then
  local cpath =
          configs.."/".. config_name
  if compare_files(
         current_config, cpath) then
    changed = true
    print("Save current config?")
    print("SET=yes, DISP=no")
    while true do
      wait_click(5000)
      if is_pressed("set") then
        newconf = true
        break
      elseif
          is_pressed("display") then
        break
      end
    end
  end
else
  newconf = true
end
```

If the folder CHDK/CONFIGS/ does not exist, we create it. If the current configuration is to be saved under a new name, we increment the configuration counter, construct a new name (conf*n*.CFG), and copy the current configuration to folder CHDK/CONFIGS/ under the new name. The user may

later want to rename this file (e.g., on a PC) and give it a more meaningful name:

```
os.mkdir(configs)
if newconf then
  ccount = ccount+1
  new_name =
         "conf"..ccount..".CFG"
  copy_file(current_config,
           configs.."/"..new_name)
  print("Saved as "..new_name)
end
```

Now we are ready to switch to the selected configuration. We update the *INI* file; if the current configuration has changed or if the user wants to switch the configuration, we copy the selected configuration file to the current configuration. So, by not selecting a different configuration file and pressing **SET**, the user can reset the current configuration to its initial state. If the user does not switch or reset the configuration, we simply update the *INI* file:

```
if selected_config then
  write_ini(
         selected_config, ccount)
  if changed or config_name ~=
               selected_config then
    copy_file(configs.."/"..
                  selected_config,
                current_config)
    print(
           "Please restart camera")
    shut_down()
  end
elseif new_name then
  write_ini(new_name, ccounend
```

This script is far from trivial, but it is developed with safety in mind. If something goes wrong, you can still retrieve a working configuration from the `.BAK` files.

Scripts like this are almost impossible to develop without a good debug environment on a PC. We will discuss this topic in the next section.

5.8 Script development

Simple scripts can be developed with a plain text editor such as *Notepad* and can be tested directly in-camera. For larger scripts, however, this can become tedious: edit the script on the PC, move the memory card to the camera, reboot the camera, run the script and find the next syntax error, move the card back to the PC, correct the error, and so on.

For larger scripts, it is much more convenient to gather some tools that allow editing within a PC environment. About 80 percent of all bugs can be caught easily while testing on the PC. After completing these tests, you will still need to do some testing in-camera—but the cycle will be much shorter.

For *uBasic,* there is a small integrated development environment (IDE) targeted at CHDK development. The program *UBDB* from *Dave Mitchell* (www.zenoshrdlu.com/kapstuff/zubdb.html) features a simple editor and a debugger. The debugger allows you to set breakpoints and step through a script line by line. Between the steps, you have the ability to change the values of parameters, variables, CHDK commands (functions), and CHDK properties, allowing you to test the script under various conditions. *UBDB* is written in Java and runs on both Windows and Mac OSX platforms.

Another comfortable editor on Windows platforms is the Open Source product *Notepad++* (http://notepad-plus.sourceforge.net/uk/site.htm). This editor is free and knows the syntax of many languages, including *uBasic* and *Lua*.

For *Lua,* several IDEs exist for different platforms. You can find a list of IDEs under http://lua-users.org/wiki/LuaIntegratedDevelopmentEnvironments.

One of them is the *SciTE IDE* that comes with the *Lua for Windows* distribution (http://luaforwindows.luaforge.net). *SciTE* integrates a debugger that allows setting breakpoints and stepping through the script line by line. However, several difficulties arise when testing CHDK *Lua* scripts under such an IDE:

- CHDK commands are not known in a PC environment and will raise errors.
- CHDK script parameters defined in the script header are not set because they are defined within a *Lua* comment block. All parameters will have the value `nil`.
- In a PC environment, *Lua* arithmetic is performed with floating point numbers. Under the CHDK, in contrast, arithmetic is performed with integers. In particular, the division operator (`/`)works differently in these environments: while 3/4 is `0.75` on a PC, it is `0` under the CHDK!

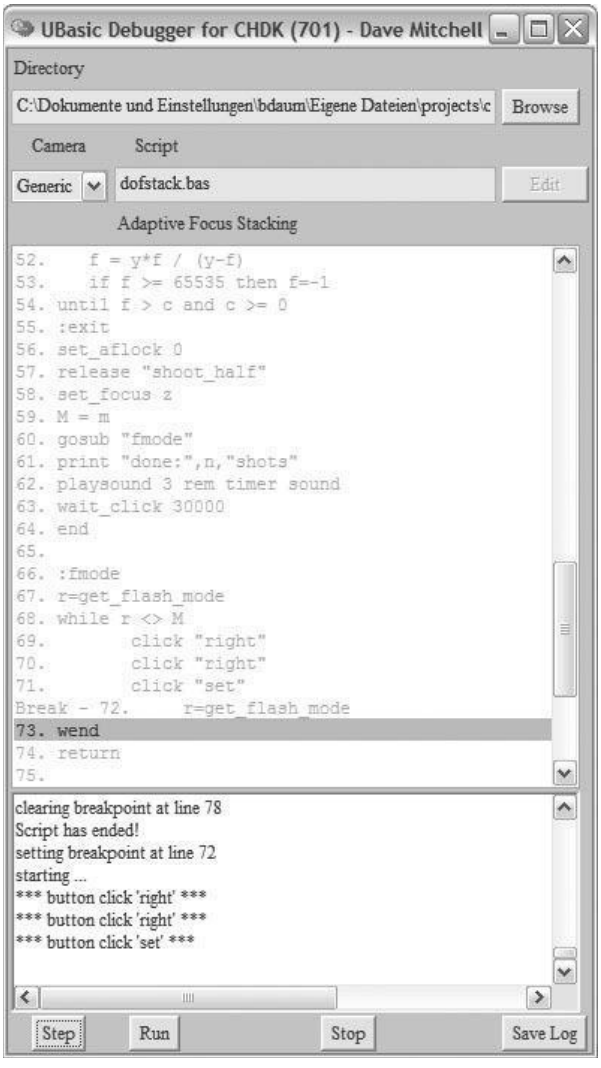


Figure 5-7
The UBDB debugger in action. Here, we step through the script from section 5.7.2. With a double click, we set a *breakpoint* on line 72. Clicking the **Step** button, we advance one more line.

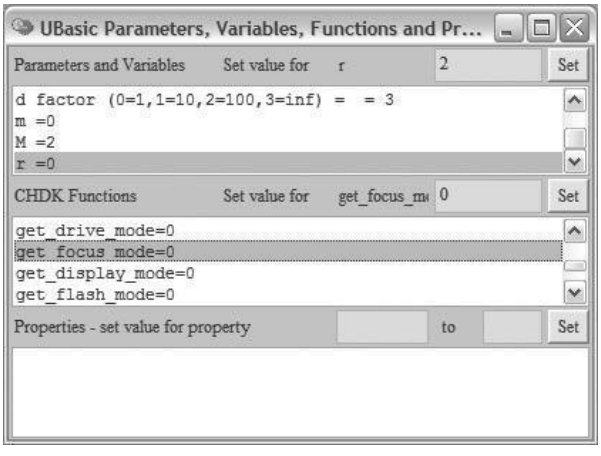


Figure 5-8
To allow the script to continue, we must now change the value of variable `r` to 2 so that the script can leave the `while` loop. This can be done in the upper section of the *Parameters* window by double clicking the variable `r`, then entering 2 into the input field (top right) and clicking **Set**. Alternatively, we can change the result value of the command `get_focus_mode` to 2 and run through the loop one more time. This can be done in the middle section of the window by double clicking the CHDK function `get_focus_mod`, then entering 2 into the input field (center right) and clicking **Set**.

However, all of these problems can be solved so that *Lua* scripts can be tested comfortably within a PC environment:

- The missing CHDK commands can be added as extra libraries to the PC environment. These libraries will only be included in the script when it runs on a PC. On the book CD, you will find these libraries in the folder `luaDebug/` under the names `chdklib.lua`, `os_ext.lua`, and `bit.lua`. Simply place these libraries into the same (PC) folder as your script. By default, library `chdklib.lua` is configured for *DryOS,* but it can easily be reconfigured for *VxWorks* (set variable `propset` to 1).
- In addition, you must include the contents of the folder `CHDK/LUALIB/` in your script folder. At the time of writing, these are the libraries `propcase.lua` and `capmode.lua`, and the subfolder is `GEN/`.
- When running a script in this environment, the parameters defined in the script header are parsed by the library `chdklib.lua` and assigned to *Lua* variables so that the script will run with the same configuration as under the CHDK.
- In case of the integer division, you should as a general rule avoid the use of the division operator (`/`) and instead use the function `idiv()`, which is defined as:

  ```
  function idiv(a,b)
        return (a-(a%b))/b
  end
  ```

  This function implements an integer division and delivers the same results on the PC and the CHDK.

All that remains to be done is to add the definition of `idiv()` and the following statement block to your script:

```
pcall(function()
          require("chdklib")
      end
)
```

When the library `chdklib.lua` is not available (as when running under the CHDK), this expression does nothing at all. The error condition raised by the `require()` command is caught by the `pcall()` function.

By using this technique, you can conveniently test a CHDK *Lua* script on a PC.

6 Advanced Techniques

In the following sections, we will discuss some advanced techniques such as panoramas and applying *High Dynamic Range* (HDR) photography to panoramas and videos. These techniques require a large number of shots to create just one end result. PC-based tools are required to create these artifacts, along with lots of patience and dedication.

6.1 Panoramas

Panorama photography is quite popular and has developed its own following within photography. The Web, in particular, has given panorama photography a new push by allowing the presentation of easily navigable 360° panoramas. There are commercial applications, too: think of real estate or tourism.

Panoramas can be created by using different techniques. They can be cut and pasted from photographic prints, like the collages popularized by the artist *David Hockney*. In analog photography, specialized panoramic cameras were used like the famous Russian *Horizon,* which has become a collector's item. In digital photography, panoramic cameras such as the 160-megapixel *Seitz* sell for the prize of a mid-sized sedan.

But special equipment is not really necessary to produce good panoramas. The stitching programs that exist today are so powerful that you can easily create your own panoramas with your Canon. All you need is a freshly loaded battery, sufficient free space on your memory card, patience, discipline, and a few tips like these:

- A tripod and a remote control (section 4.9) are recommended, but I have shot acceptable panoramas hand-held. If you use a tripod, you should also use a spirit level to align it perfectly with the horizon.
- Zoom out as much as you can. A wide-angle lens position allows you to do a panorama with only a few exposures. You may want to put your camera intro portrait orientation, especially if you are doing hand-held panoramas—you have to consider the later trimming.
- To avoid later registration problems, you should pivot the camera around the *nodal point,* which basically is the center of the lens. But if there are no subjects at close range, this does not matter as much.

Precise turning can be achieved with a *nodal point adapter* mounted between tripod and camera. If you want to do multi-row work (for spherical panoramas), you need a *VR head*. Several manufacturers offer such devices; some of them are even motorized. If you use one of these devices, you must determine the nodal point of your camera at the focal length that you're using (because the nodal point moves when zooming).

To do so, first mount the camera on the nodal point adapter and the tripod. The adapter allows you to shift the camera forwards and backwards. Set up the camera and tripod so that you have both a far and a near object in the viewfinder. When you turn the camera, the distance on the screen between both objects must not change. To judge this distance precisely, you can make a picture and magnify it using the zoom rocker in replay mode. Should the distance between the two objects change while turning the camera, shift the camera forwards or backwards until you find the optimal position. Mark this position for the next time.

- Zoom level and focus should be left unchanged for all exposures. For landscape panoramas, setting the camera to infinity is fine in most cases. Otherwise, you need to use the *Override* functions (section 4.3.1) to dial in a fixed subject distance. A varying subject distance like that caused by the autofocus system would make it difficult for the stitching program to join the single exposures.
- Exposure is often a problem with panoramas, especially on a bright day. The contrasts can be very high because you cover a large area. Some photographers use a manually set exposure and keep it constant through all of the images. In most cases, this will result in overexposed and underexposed areas. If you want to work in this way, make sure that you get the highlights right and let the shadows drown.

  Personally, I would rather use automatic exposure for each picture so that the camera can adapt to different lighting conditions. Modern stitching software can detect these differences (it reads the exposure data from the EXIF data) and blend the images seamlessly together. Especially with high-contrast scenes, the results are usually better. Some stitching software is even able to store the panorama in the form of a HDR file to keep the full contrast range for later tone mapping (section 4.6.2).

  What you should do, however, is use the same ISO value for all images. This will give you a uniform noise level over the whole panorama.
- Do not use automatic white balance but rather set the camera to a fixed color temperature, e.g., to *Daylight* or *Cloudy*.
- When turning the camera, allow for sufficient overlapping between the single images. About one third of each image should overlap with the previous image. The CHDK offers a great tool to register the single

images precisely, especially when shooting panoramas hand-held: *Edge overlay* shows the edges of the previous image on top of the current viewfinder content (section 4.7 and Figure 4-40).

- Stitching programs usually accept TIFF and JPEG files. Shooting JPEG is just fine. If you want to use RAW files for the ultimate quality, make sure you have enough space on your memory card. Also make sure to develop each RAW file with exactly the same parameters.

After you create a series of panoramic shots, you will need to stitch them together. There are several programs that can help you in that task. These programs differ in details, quality, and level of manual control but more or less keep to the same process:

1. The program searches for prominent points within the single images (using the SIFT algorithm or a similar algorithm)—points that clearly stand out and can be used for registering images.
2. The next step is to match the points from the single images. This is usually done with the RANSAC algorithm. Most of the stitching programs are able to accept an unordered set of images and will put them into the right sequence.
3. The single images are shifted, rotated, and stretched to make them fit together.
4. The single images are further corrected by removing the effects of vignetting (lens shading), barrel and pincushion distortions, and different exposures.
5. The single images are fused together. This happens within the overlapping area by choosing a (usually irregular) borderline that is not too obtrusive. The transitions are softened up, and ghosts within the transition area are masked out.

There are many commercial panorama stitchers but also some very capable free programs:

- First, *Photoshop* offers *Photomerge,* an easy-to-use panorama stitcher. It's a tool for the quick panorama without too many configuration options. It's free, too, as long as you own a Photoshop license.
- The *Microsoft Image Composite Editor* (ICE) is similarly easy to use, and it's completely free.
- *Autopano Pro* from *Kolor* is a very powerful panorama stitcher. Panoramas can be created automatically, but you also have full manual control over each step. The program can keep the whole contrast range delivered by the single images and allows storing the panorama as an HDR file. It can even combine several HDR stacks, resulting in true HDR panoramas (section 4.6.2).

- The same features in Autopano Pro are available in *PTGui*—one of the pioneers of panorama technology.
- *Hugin,* which doesn't lack in power, is the free Open Source alternative for panorama stitchers. It supports HDR panorama stitching and can also be used for nonpanoramic purposes, such as perspective correction of architectural shots.

All panorama stitchers allow you to choose among different projections. A projection is responsible for how the image pixels are mapped onto the viewing surface—i.e., how they are projected. All programs support at least rectilinear, cylindrical, and spherical projection. Some of them support many more and even rather exotic projections.

- The *rectilinear projection* simply projects the whole panorama onto a flat surface. Because this leads to strong distortions towards the edges of the image (similar to a photo taken with an ultra-wide-angle lens), the maximum angle of field should be limited to 120°.
- The *cylindrical projection* projects the panorama onto the inside of a cylinder that is then unrolled onto the viewing surface. This projection and its variants (e.g., *Miller projection*) are best suited when you want to print panoramas with a wide angle of field.
- The *spherical* or *equirectilinear projection* projects the panorama onto the inside of a sphere that is then unrolled onto the viewing surface. This projection is typically used for spherical panoramas viewed on a computer display using interactive viewers.

Viewers for spherical panoramas are available as a *Java* applet or as a *Flash* application. *Apple Quicktime* can show panoramas, too. Among the *Java*-based viewers, the free *PTViewer* is interesting. It can handle cylindrical and spherical projections and also display true HDR panoramas (see next section) with the full contrast range. When the user changes the angle of view, the viewer will adapt to the brightness of the scene, much as the human eye would do. For *Flash,* there are many commercial offerings for panorama viewers plus the notable exception of the *panoSalado* viewer that comes for free.

6.2 HDR Panoramas

We have already mentioned one of the problems in panorama photography: the large scene contrast. When the sun is shining and the angle of view is 180° or greater, you will have images that are shot with the sun in the back and images shot straight into the sun. But contrast at night can also be too high for a classical exposure.

In section 4.6.2 we already discussed how to create an HDR series with the help of the CHDK. Basically, you can use the same technique for each single image of the panorama:

- First, go to ALT > MENU > *Extra Photo Operations* and select **Off** for *Disable Overrides*.
- Then go to ALT > MENU > *Extra Photo Operations* > *Bracketing in Continuous Mode* and dial in a *TV Bracketing Value*. For normal bracketing work, a good value is **2 Ev**, but for panorama work a bit less will make the registration of images easier. As *Bracketing type* choose **+/–**, a typical set-up for HDR stacks.
- Finally, switch the camera's shooting mode to *Custom Timer* and configure the timer with a *delay* of **0** sec and the number of *shots* set to **3** or **5**, depending on the contrast in the scene.
- Now, when you press the shutter button, you should get one correctly exposed picture, one overexposed picture, one underexposed picture, and so on.

After you set up the camera in this way, you can take the images for the panorama (don't forget to set a fixed ISO value and a fixed white balance). Press the shutter button, wait for the images to be taken, turn the camera, press the shutter button again, etc. A tripod is recommended but not absolutely necessary. Make sure to use fresh batteries and a memory card with enough free space.

Let's assume that the focal length of your camera lens is equivalent to 35mm at a wide-angle setting. If you're using the camera in portrait mode with an overlap of one third between each panorama part, you can cover an angle of view of approximately 25 degrees. For a single row panorama, this would result in 15 parts of three images each, or 45 total images. Even if you shoot JPEG, you would end up with approximately 120 MB. Shooting RAW would fill your memory card with approximately 450 MB! Multiple row panoramas, such as those required for 360° spheres, take a multiple of that value.

Combining the image series into one HDR panorama can be performed with a suitable panorama stitcher. *Autopano Pro, PtGui Pro,* and *Hugin* can all compose HDR panoramas. The output can be saved in an HDR file format such as *OpenEXR* or *HDR Radiance*. The resulting file can then be tone mapped using an HDR composer (e.g., *Picturenaut, Photomatix Pro, FDRTools Advanced,* or *Dynamic Photo HDR*) to produce printable HDR output.

Example project

The following image (Figure 6-1) demonstrates the use of panorama techniques to solve a problem often encountered by owners of compact

cameras: the lack of interchangeable lenses. Most compact cameras allow for a wide-angle zoom position of 35mm focal length (35mm equivalent), some even down to 28 mm. But when compared with the ultra-wide-angle lenses available for DSLR owners (24 mm down to 12 mm equiv.), one can only feel envy.

With panorama techniques, you can simulate an ultra-wide-angle lens. Instead of using a cylindrical projection, you use an equilinear projection when stitching the images. The image in Figure 6-1 consists of nine single shots, three shots in three rows. The camera was a *Canon Digital Elph SD1100* at a wide-angle setting (equiv. 38 mm). Because the contrast range was rather high, I decided to use HDR bracketing as well, with three images per shot, and a *Tv Bracketing value* of 2 EV. This resulted in 27 single images.

In such a case, you should consider selecting the next lower image resolution. Some HDR tone mappers cannot handle images that are too large, and processing times can become excessive. The images delivered by the panoramic stitcher can be very large, indeed! Because I had taken the images at full resolution, I scaled them down afterwards to 1600x1200 pixels. The images were then composed in *Hugin* with virtually no manual adjustments. The only thing I had to do was supply a format factor of six manually. (*Hugin* proposed a wrong format factor because of the extra scaling process.) Also, I set the projection to *rectilinear,* the exposure optimization to HDR, and the output format to HDR. The resulting image had a horizontal angle of 125° (145° in the diagonal). The simulated focal length was 7 mm (35mm equivalent), a value DSLR owners can only dream of. The resulting EXR file was then opened with *Picturenaut* and tone mapped using the *bilateral* method.

Figure 6-1
Breuburg Castle, Germany. Even after some heavy cropping, the image still has a horizontal angle of 88° (98° in the diagonal).

6.3 HDR videos

Some users have created HDR time-lapse videos with their Canons. The technique is quite simple: set the camera to bracketing mode and configure the *Custom Timer* as discussed in section 6.2. Then use any time-lapse script that can press the shutter button in defined intervals, such as the script developed in section 5.7.1. Again, be sure to have fresh batteries and rather large memory cards on hand (see appendix A.1 for running the CHDK with memory cards larger than 4 GB).

The resulting bracketing series must be processed with an HDR composer. The composer should be able to combine images in batch mode[1]. Make sure that tone mapping is performed with the same parameters for each bracketing series. The result should be stored in the form of JPEGs that are then combined into a movie with a time-lapse converter (section 5.7.1).

1 *Photomatix Pro, FDRTools Advanced, Dynamic Photo HDR* all have batch facilities.

7 The Stereo Data Maker (SDM)

In the past, finding the right CHDK version was sometimes a disturbing experience because of the many different builds (identified by numbers) and spin-offs such as the *Allbest build, Fingalo's build,* and *Microfunguy's SDM build*. Different people had different ideas, wanted to try them out, and therefore created spin-off versions of the CHDK. By now, most of this is history. The new functions implemented in the various spin-offs have been imported into the main CHDK trunk (called the *Morebest build*), so we have one unified release. Almost, that is.

The SDM *(Stereo Data Maker)* has survived and has its own community of users. The SDM is a lightweight version of the CHDK, removing some rarely used functionality but adding other functions. As the name suggests, it provides additional functionality for stereo (3D) photography such as camera synchronization. But in fact, the SDM is not used dominantly for that kind of photography. Because of its excellent remote control features and its integration with external devices, it is quite popular in the KAP *(Kite Aerial Photography)* community.

7.1 Installing the SDM

The SDM can be downloaded from http://stereo.jpn.org/eng/sdm/index.htm. The website lists the models and firmware versions that are supported—not as many as the original CHDK, but still quite a number. Your first step should be to determine whether your camera is supported by the SDM and what the firmware version of your camera is. We have already shown how this is done in chapter 3.

Afterwards, download the file `common_files.zip`. Unpack this file into a suitable folder on your computer. You can now use the program `sdminste.exe` found in the ZIP file to install the SDM. This *SDM Installer* is capabable of formatting your card and making it bootable but will only run under *Windows*. If you are using a different operating system, please refer to section 3.4 on how to create a bootable card without the help of a PC-based program.

You should, of course, first backup your memory card. Afterwards, switch the write protection off and insert the card into a card reader. Make sure that there is no other removable media online so that it's not accidentally

reformatted during the following process. Then start `sdminste.exe` and format the card with FAT, not FAT32 (assuming that your card has no more than 4 GB capacity). After formatting the card successfully, click the button *Make bootable Disk*.

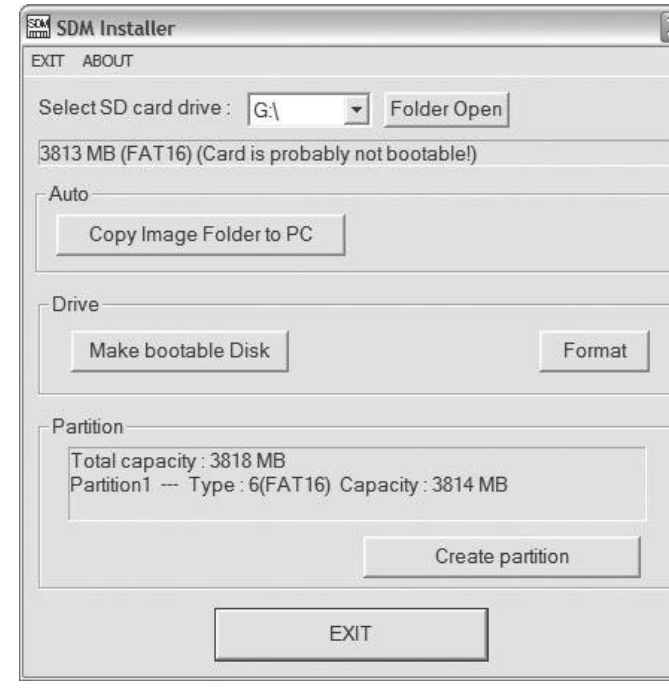


Figure 7-1
The SDM Installer in action

Now you can transfer the folders `CHDK/` and `DCIM/` from the SDM deployment onto the card. These folders contain the language files (English, Spanish, French, Hungarian, and German), several predefined grids (section 4.2.6), a font, and some useful scripts (motion detection, remote control, time lapse). If you need a larger variety of fonts, you can use the fonts deployed with the "normal" CHDK (they are compatible) and place them into the folder `CHDK/FONTS/`.

The SDM deployment also contains a folder with predefined SDM configurations. These configurations are meant to help you set up a pair of cameras for stereo photography:

- If your cameras will be mounted on a *Z-frame* (section 7.7.4), copy file `CONFIGS/config_eng/L/CHDK.CFG` into folder `CHDK/` on the memory card used for the left camera. The file `CONFIGS/config_eng/R/CHDK.CFG` goes into folder `CHDK/` on the memory card for the right camera.
- If your cameras will be mounted on a *U-frame* (section 7.7.4), copy file `CONFIGS/config_eng/L_horizontal/CHDK.CFG` into folder `CHDK/` on the memory card used for the left camera. The file `CONFIGS/config_eng/R/CHDK.CFG` goes into folder `CHDK/` on the memory card for the right camera.

It's a good idea to mark your cards with a large "L" and a large "R" respectively.

After these common files are installed, it's time to download the camera and firmware-specific SDM core from http://stereo.jpn.org/eng/sdm/index.htm. This download contains a single file, `DISKBOOT.BIN`. Copy this file into the root directory of your memory card.

Now, remove the card from the card reader, switch on the write-protection, and insert the card into the camera. When you start the camera, you should see the SDM splash screen.

7.2 Restrictions

We have already mentioned that the SDM is based on a lightweight CHDK variant. Some CHDK functions have been removed. In many cases, these functions are not really needed for photography. For example, the calendar and the four games have been omitted from the SDM.

Unfortunately, there is also true photographic functionality that is missing—and your only option is to go back to the CHDK if you require that functionality. For example, the SDM has only limited RAW support. In particular, it cannot create DNG files (section 4.5.2). In-camera development of RAW files is missing, too, and curves are not supported.

In regard to scripting, the SDM supports its own *uBasic* dialect (section 7.9). Because some CHDK *uBasic* commands are not supported by the SDM, not every script can be ported to the SDM. Scripting with *Lua* is not supported at all.

7.3 Additional functions

On the other hand, the SDM introduces so many useful functions that you might want to switch to the SDM now and then. It might be a good idea to keep two memory cards, with the CHDK on one and the SDM on the other.

In particular, the SDM improves *onion skinning* (overlaying two pictures) by allowing overlays to be viewed as red/cyan *anaglyphs*. This is possible both in color and black-and-white. Of course, this feature is designed for stereo photography, but it can also be useful for controlling panorama or bracketing series. Simply go to *Replay mode,* half-press the shutter button, move to the next image with the RIGHT button, half-press the shutter button again, and you will see both images either side-by-side or overlaid as a red/cyan anaglyph, depending on your settings.

Time-lapsing is also made easy with a single, powerful *uBasic* command that allows you to create highly sophisticated time series. A script version of that command is already contained in the deployment so it can be

readily executed. This feature and the excellent *remote features* (including support for the *Ricoh CA1* remote control) are the reasons why the SDM has so many users in the KAP community. However, recent versions of the CHDK have adopted the SDM remote functions and offer almost the same functionality.

A specialty of the SDM is the use of the blue LED as a *serial interface* to hook up external devices, such as motorized camera rigs or robotic panorama heads. Finally, a recently introduced feature of the SDM is the support of *Digiscoping,* i.e., connecting a camera to a telescope.

7.4 Operation

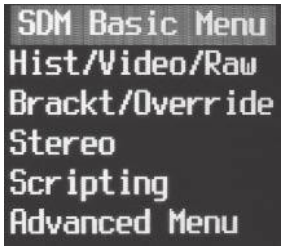


Figure 7-2
The SDM main menu looks different compared to the CHDK menu because the already well-known functions are grouped differently.

Using the camera under the SDM is quite similar to using it under the CHDK. There is an *<Alt>* mode which you can enable via the ALT button (chapter 4). Pressing the MENU button in *<Alt>* mode leads to the SDM menu.

The various indicators are also organized differently. There is an SDM header line showing several entries, such as shooting mode, zoom value, and focal distance. You can configure this line under ALT > MENU > *Stereo* > *SDM Header*.

Many functions are invoked through key shortcuts instead of menu entries. For example, enabling or disabling *Overrides* is performed by half-pressing the shutter button and pressing FUNC/SET. An indicator at the left side of the display shows **MAN** when *Overrides* are enabled, and **OFF** when *Overrides* are disabled.

Here are some more key shortcuts. They are worth remembering because they can make life easy. However, you can always look them up with the *Text File Reader* (ALT > MENU > *Advanced Menu* > *Text file reader* > *Open new file...*). File CHDK/TEXTS/SHORTCUT.TXT lists all of these shortcuts.

In *Record mode,* the following key shortcuts are available:

- ALT-MENU: Invokes the SDM menu.
- ALT > FUNC/SET: Switches the OSD on/off with a short click on FUNC/SET. A longer press switches the *Edge Overlay* on/off.
  Note: This key combination is used in standard CHDK for loading a script. Under the SDM, you must use the menu function ALT > MENU > *Scripting* > *Load script from file...,* instead.
- ALT > LEFT: Browses OSD information screens backwards. These screens display vital data for different purposes such as digiscoping or stereo set-up organized in multiple pages.
- ALT > RIGHT: Browses OSD information screens forwards.

If the camera is set to manual focusing with **ALT** > **MENU** > *Brackt/Override* > *Focus mode* to **Manual**, the above shortcuts are interpreted differently[1]:

- **ALT** > **LEFT**: Focuses nearer
- **ALT** > **RIGHT**: Focuses farther
- **ALT** > **UP**: Infinity
- **ALT** > **DOWN**: Hyperfocal distance

When focusing with these keys, observe the SDM header line to see how the focal distance changes. The changes become effective immediately, and you can observe how the area of sharpness changes on the display. However, the dialed-in focal distance is only used in the next shot if the *Overrides* have been enabled (see above). Otherwise, the camera will refocus.

In *Play mode,* the following key shortcuts are possible:

- **ALT** > **FUNC/SET:** Switches *Remote Browsing* on/off with a short click on **FUNC/SET**. A longer press switches the *USB Upload* function on/off.
- **ALT** > **LEFT**: Browses OSD backwards.
- **ALT** > **RIGHT**: Browses OSD forwards.
- **Half-press shutter button**: Shows red/cyan overlay of left and right image. On the S2 IS, S3 IS, S5 IS, and A550, use the **FUNC/SET** button instead (section 7.6).

There are a few more shortcuts listed in file `SHORTCUT.TXT` that apply to specific situations. Don't forget to press the **ALT** button again after using a command sequence starting with **ALT**. Otherwise, you will still be in *<Alt>* mode and your next shutter button press will probably start a script instead of take a picture.

7.5 Remote control

In contrast to the CHDK, the SDM switches on the USB remote control by default. All you need to do is plug in the remote control unit. Two different remote systems are supported:

- The *Ricoh CA1* remote control and compatible models. The switch of this unit works in three stages: (1) When the button is half-pressed, the camera focuses. (2) When the button is fully pressed, the camera measures the exposure. The blue LED indicator is lit when the camera is

1 Switching between these two modes requires at least a half-press of the shutter button after the focus mode is changed.

ready to shoot. (3) When the button is released, the camera fires. It is possible to combine steps (1) and (2) by full-pressing the button once.

- A simple *switch-and-battery* remote control as shown in section 4.9.2. (1) Press the button for less than half a second to focus. (2) Press the button again to determine the exposure. The blue LED indicator will be lit when the camera is ready to shoot. (3) Release the button to shoot. It is possible to combine steps (1) and (2) by pressing the button for longer than half a second, waiting on the blue LED, then releasing the button.

This mode of operation is true for the "normal" camera mode, also called the **DIRECT** mode (you may have wondered what the **DIRECT** indicator in the SDM header line means). In this mode, a photo is taken when the switch is released.

In **FAST** mode, in contrast, a photo is taken as soon as the switch is pressed. The **FAST** mode is entered automatically when ALT > MENU > *Brackt/Override* > *Focus mode* is set to **Manual** or to **Digiscop**.

The remote system can be configured with the same parameters as shown in section 4.9.3. Since this functionality was adopted by the CHDK from the SDM, there are almost no differences between the versions. The menu is structured differently, and the names of some menu items are different. For example, the CHDK option *Enable Remote Zoom* is called *Synch Zoom* in the SDM and is located in the *Stereo* submenu. The reason is obvious: controlling the zoom function through the remote control allows for synchronized zoom operation when two cameras are connected to the remote control. This is an ideal way to set up twin cameras for stereo photography.

The other remote parameters are located in the submenu *Stereo* > *Synchronization*. There is one more parameter that is not present in the CHDK: *Add User and flash delays*. This option can be enabled to facilitate synchronized flash (section 7.7.5). For each camera, a special delay value (depending on the shutter speed) is computed and added to the delay values dialed in by the user.

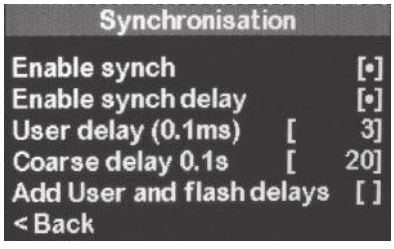


Figure 7-3
The SDM *Synchronization* submenu. The functionality is nearly the same as in the CHDK, only the SDM allows for adding extra flash delays.

7.6 Communications

In this section, we will discuss the camera uplink to a PC as well as serial communication with an external device.

7.6.1 USB upload

Because the USB port, by default, is reserved for the remote control, you must explicitly **disable** the option ALT > MENU > *Scripting* > *Disable USB Download* if you want to upload (or download, depending on your angle of view) images onto a PC. Of course, the camera must be switched to *Replay mode* when uploading images.

The SDM upload functions are designed to work closely together with the WIA Loader (www.mortara.org/index.php/software/windows/49-wia-loader), a *Windows* (.NET) based image uploader. (On Linux and MacOS X, it can run under the *Mono*-Framework 2.4.) The WIA Loader can, for example, take care of stereo image pairs from twin cameras (section 7.7.4). It can rename such images, place them in left and right folders, and rotate images losslessly if one camera is mounted upside down in a Z-frame.

Because transmission takes place through the USB port, the card can remain in the camera. This has advantages in that the camera doesn't need to be removed from an elaborate set-up and that the card is never forgotten in the computer. The WIA loader can even automatically remove transmitted images from the card. Because the SDM allows uploading images via script commands (section 7.9), you can (almost) simulate tethered shooting—at least for time-lapse operation. After an image has been taken, the interval between two pictures can be used to upload the image to the connected PC. And because the WIA loader can also delete the image from the card, it never runs out of capacity.

7.6.2 Serial communications

Not only can the SDM communicate with a PC, it also offers the possibility of using the blue LED as an optical serial port. A light-sensitive cell glued to this LED and connected to a microcontroller (or to the sound card of a PC) is all the hardware required for receiving data from the camera. A set of script commands (section 7.9) is available to initialize the interface and to send data to the connected external device. The device can even acknowledge the successful reception by sending a signal to the V+ pin (supply voltage) of the camera's USB port (ALT > MENU > *Scripting* > *Disable USB Download* must be **enabled**). This is not absolutely necessary but leads to more reliable communication.

Transmission begins by sending a start bit with a pulse width of 4. 24 data bits (3 bytes). A 0-bit is encoded with a pulse width of 1, a 1-bit with a pulse width of 2. The space between two bits has the length of one pulse. After the bit sequence, a space with a pulse width of 3 follows. Then the above sequence is repeated. The receiving device should compare both sequences, and in the case of equality, acknowledge successful reception at the USB port. In the case of inequality, no action needs to be taken—the SDM command for receipt at the USB port will simply time out.

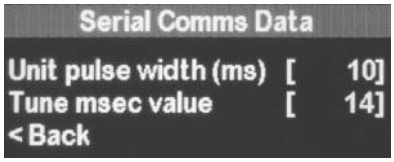


Figure 7-4
The *Serial Comms Data* submenu controls the pulse width for serial communications via the blue LED.

The pulse width can be set in **ALT** > **MENU** > *Advanced* > *Serial Comms* and should be between 10 msec and 20 msec. At the beginning of a communication, a script will typically send a reference pulse (command `unit_pulse`) that allows the external device to calibrate to that pulse width. Because the real pulse width differs from the specified value (depending on the camera's processor), the menu offers the ability to fine-tune the pulse width. This is done by trial and error, or by analyzing the signals with an oscilloscope[2].

This serial communication facility is typically used for controlling external devices with a camera script, such as robotic camera platforms or rigs. An interesting project for building a DIY panoramic robot head is found on www.instructables.com/id/Camera-Panorama-robot-head-panograph/.

Of course, it is also possible that an external device will send more data to the camera's USB port than a simple receipt. Such data can be encoded with pulses of varying width and interpreted with the help of the script command `get_usb_power` (section 5.5.10).

7.7 Stereo photography

The SDM was designed for stereo photography. When we look at nature to see how 3D vision works, we soon find that in almost every case two images taken at different positions are compared with each other. For example, humans have two eyes set at a distance of approximately 70 mm apart. Our eyes compare images simultaneously, allowing us to perceive movement in space very well.

Other animals have eyes that are oriented sidewise. In that case, it is often impossible to compare the images from both eyes simultaneously because each eye sees a different part of the scene. Instead, such animals compare images obtained before and after a movement. Take, for example, a chicken: a first look, a quick move of the head, then a second look. The faster the head moves, the better the comparison between images works. Now you know why a chicken walks like a chicken.

2 The *Visual Analyzer* from www.sillanumsoft.org teamed with the sound card of your computer makes a capable and free oscilloscope.

7.7.1 Stereo photography with a single camera

What works for a chicken can work for us, too. It isn't really necessary to use two cameras for creating stereo images. If the subject matter doesn't move, you can obtain good results by using a single camera: take the first shot, move the camera a bit to the right, and take the second shot. This method is particularly suitable for macro stereo photography. In macro photography, the shift between the left and the right image must be quite small—so small that it's impossible to mount two cameras side by side without exceeding the optimal distance between the two lens centers.

Shifting a single camera, however, sounds easier than it is. It can be quite hard to make both images register correctly, especially when shooting hand-held. The SDM offers (just like the CHDK) the *Edge Overlay* as an aid:

- First, switch on the *Edge Overlay* by clicking the **ALT** button and then pressing the **FUNC/SET** button for a second. You should now see a notice such as "Allocated xxxxxx bytes". Press **ALT** again to leave the *<Alt>* mode.
- Press the shutter button a first time. The *Edge Overlay* should appear now in the form of bright yellow lines. Also, the message "frozen" is shown.
- Move the camera a bit to the right, but not too much. Use the distance between your eyes as a guide and also see below. For macro work you can choose a much smaller distance, and for landscape work you can exaggerate the 3D impression by choosing a larger distance.
- Now register the camera so that the edges of the nearest subjects match. The difference between the edges of the farthest objects *(deviation)* should not exceed two millimeters on the screen (assuming a display width of approximately two inches).
- Take the second shot.
- Now switch to *Replay mode*. Navigate to the first (left) image and half-press the shutter button[3] to select it. Then navigate to the second (right) image and half-press the shutter button again. The display should now show a red/cyan overlay of both images *(anaglyph)*. If you have your red/cyan goggles with you, you can view the image in 3D right away. Under **ALT** > **MENU** > *Stereo* > *Playback mode* you can choose between a monochrome anaglyph, a color anaglyph, or a side-by-side pair.

  The main purpose of this display option is, however, to check the registration of both images. If you want to magnify the anaglyph, you must first zoom in on the left image, half-press the shutter button, then

3 On the S2 IS, S3 IS, S5 IS, and A550, use **FUNC/SET** instead of the shutter button half-press.

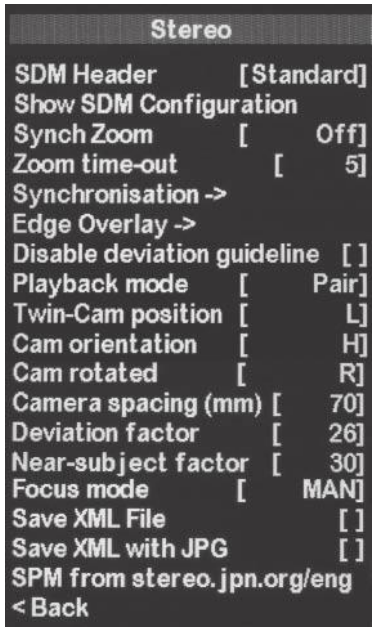


Figure 7-5
The SDM *Stereo* submenu combines the settings relevant to stereo photography. It includes submenus for *Camera Synchronization* (Figure 7-3) and *Edge Overlay*.

Figure 7-6
The Single Camera OSD allows capturing both far and near distance, and computes a shift value that must be entered under *Deviation factor* in the *Stereo* submenu. Here, the value is too small (the minimum deviation factor is 10) because the lens is still in telephoto position. Zooming out will make the "Shift" value bigger.

zoom out, navigate to the right image, zoom in again, and half-press the shutter button. Working in this way, you can control the registration of images very precisely.

Note: this display function does not work on all cameras. On some cameras, the odd/even position of the images in the browsing sequence can prevent the creation of the anaglyph. If this is the case, you should shoot a spare image before shooting the stereo pair. If the anaglyph creation fails, you can delete the spare image and try again.

While the above method works well for mid-range shots, a closer analysis is required for close-up work. If you register both shots for a very close object, the deviation for far objects may become too large. (Above, we allowed for a maximum on-screen deviation of 2 mm.) When you look at the composite image later, your brain may be unable to cope with such a large deviation and both images will fall apart.

To help you with the problem of determining the right deviation, the SDM offers a tool for computing the maximum acceptable deviation between the positions of the farthest objects. The following method is based on measuring the exact distance of near and far objects. This is done at the telephoto position of the zoom lens in order to obtain precise readings. From the measured distances, the maximum deviation value is computed and visualized on the display.

- Set the camera's *AF mode* to **Center**.
- Under ALT > MENU > *Stereo* > *Camera Spacing*, dial in the distance that you want the camera to shift between both pictures. Typical values here are 60–70 mm for mid-range work and 10–20 mm for close-up work.
- *In the same menu, set the Twin*-Cam position to **L** (left) and the *Cam orientation* to **H** (horizontal) or **V** (vertical), depending on the camera orientation.
- By repeatedly pressing ALT > LEFT or ALT > RIGHT, browse the *SDM OSD pages* until you reach a screen entitled "SINGLE CAMERA". Press ALT again to leave the *<Alt>* mode.
- Zoom to the longest telephoto position (do not use digital zoom). This is mandatory! Focus on the nearest object and half-press the shutter button. The entry "Near" in the OSD screen will be updated. It will turn red if the subject distance is below the *near limit*. This near limit is a result of multiplying the values dialed in under *Near-subject factor*[4] and *Camera Spacing*. The *Near-subject factor* specifies how much perspective distortion is acceptable to you when the camera viewpoint changes (Figure 7-7). The smaller this value, the better the quality of the compound image, but the larger the near limit will be.

4 Typical values are 30 for mid-range work and 15 for close-up work.

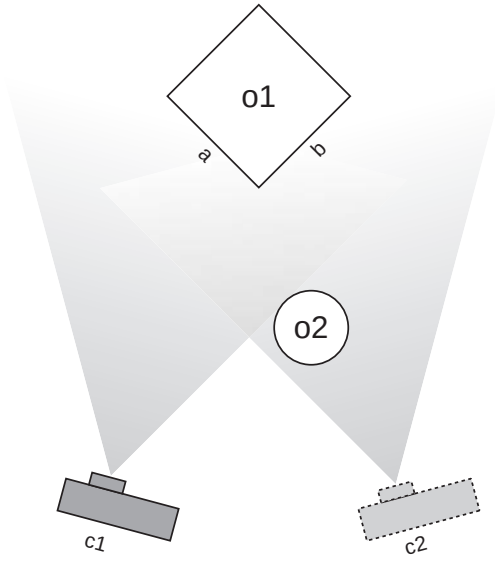
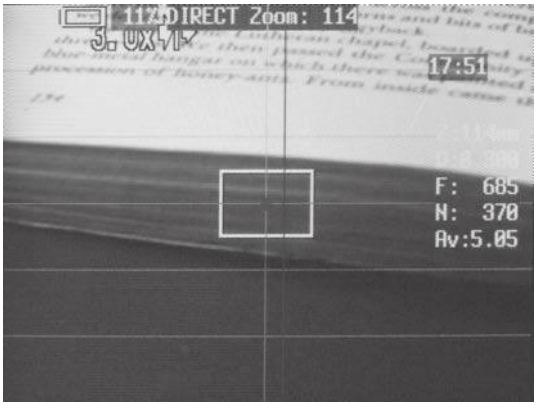
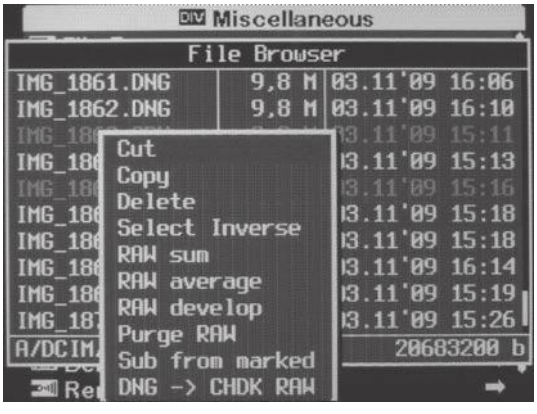


Figure 7-7
Close-up work and stereo photography have their own problems. Here are two: camera c1 sees object o1 from a different perspective than camera c2. c1 sees more of edge a while c2 sees more of edge b. Object o2 is so close that it is only seen by camera c2. It will appear as a ghost. In both cases, reducing the distance between the cameras or increasing the subject distance will help.

- Still in telephoto position, focus on the farthest object and half-press the shutter button. The entry "Far" will be updated.
- Zoom back to the desired zoom position. The value of entry "Shift" will change with the focal length.
- Dial in the resulting "Shift" value under *Deviation factor*. The image pixel width divided by the *Deviation factor* determines the maximum deviation in pixels for later viewing.
- Now you are ready to display the guidelines. First, disable the option *Disable deviation guideline*. Also set the *Twin-Cam position* to **R**.
- Then go to **ALT** > **MENU** > *Advanced Menu* > *OSD parameters* > *Grid settings*. Enable the option *Show grid lines* and invoke the function *Load grid from file...* to load `Deviation_H.GRD` (`DEVIAT~1.GRD` on DryOS).

Figure 7-8
The grid `Deviation_H.GRD` in action. The horizontal lines can be used for orientation and alignment. The distance between the vertical line in the center and the deviation line at the right indicates the maximum acceptable deviation.





- You can now use the *Edge Overlay* to register the nearest objects. The *Deviation Line* at the right-hand side of the grid's centerline shows the maximum deviation that is allowed for far objects. If this distance is exceeded, you should reduce the camera spacing and redo the above steps.

Unfortunately, this method does not work in all cases. Especially in macro mode, your camera might not be able to focus at close distance while the lens is in telephoto position. The SDM, on the other hand, can only capture distances in telephoto position.

In this case, you are thrown back to using some old, well-known rules for computing *Camera Spacing* and *Deviation factor:*

- For close-up work, the *Camera Spacing* should be lower than the distance between the lens and subject divided by 20. So, when your subject matter is 30 cm away from the lens, you should use a *Camera Spacing* of less than 15 mm.
- For "normal" subject distances, the *Camera Spacing* should be lower than the distance to the nearest point of the subject divided by the *EFL* (35mm equivalent focal length) plus 20%. For example, if the distance to the nearest point is 3 m, and the current EFL is set to 50 mm, an acceptable *Camera Spacing* would be 3000 / 50 + 20% = 72 mm.
- The *Deviation factor* should be, in most cases, between 25 and 30. The lower the value is, the larger the maximum acceptable deviation `dmax` will be, and the more difficult the viewing experience will be. The maximum acceptable deviation is computed by dividing the image width by the deviation factor. After entering this factor, you can control the actual deviation with the grid `Deviation_H.GRD` and the *Deviation Line* as described above.

If you want to play safe, you can "bracket" the *Camera Spacing,* e.g., by following the "thirds" method (Figure 7-9). Later, when combining the images, you can select the best pair for achieving the desired 3D effect.

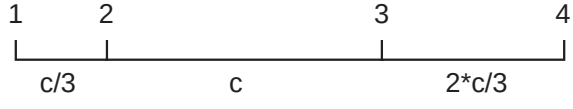


Figure 7-9
Given the computed camera spacing c, four shots are sufficient for covering a stereo basis between c/3 and 2*c. By selecting two shots from this series, you can decide between c/3 (1+2), 2*c/3 (3+4), c (2+3), 4*c/3 (1+3), 5*c/3 (2+4), and 2*c (1+4).

7.7.2 Producing and viewing composite stereo images

Once you have created pairs of stereo photos, you may want to compose them into a single file so that they can easily be deployed and viewed. One program that can do exactly that is *StereoPhotoMaker,* which is available for free on the Web (http://stereo.jpn.org/eng/stphmkr). If you are interested in stereo photography, it is also worth visiting the parent page (http://stereo.jpn.org/eng) for even more tips and tools.

Simply download the ZIP file and extract it into a folder of your choice. To align the stereo pairs automatically, the *StereoPhotoMaker* needs an additional component[5], the free program *Autopano* from http://autopano.kolor.com/. Unzip the downloaded ZIP file and copy `autopano.exe` into the installation folder of the *StereoPhotoMaker*. This program is good enough to correctly align images that are shot hand-held.

After installation, simply start the *StereoPhotoMaker* with a double-click. Then you can drop a pair of stereo images into the image area, invoke *Adjust* > *Auto align,* and then invoke one of the *Stereo* > *Gray Anaglyph* or *Stereo* > *Color Anaglyph* functions. Press **Enter** for full screen view, turn down the room light, put on the red/cyan 3D goggles[6], and your computer screen becomes 3D. With *File>Save Stereo image,* you can save the anaglyph to a new file that can be viewed with any image viewer. *StereoPhotoMaker* can, of course, do much, much more, but this is the shortest way to your first 3D photo. Not only red/cyan goggles are supported, but also 3D shutter glasses that synchronize your view with quickly alternating left/right images on the screen. Another option for producing 3D images is the *AnaglyphMaker* from www.stereoeye.jp/software/index_e.html.

7.7.3 Stereo focus stacking

A problem with close-up stereo photography is the narrow depth of field. In contrast, when you view a scene, your eyes scan it and adapt to the varying distances. Your brain does the composition.

We already know that we can do something similar with focus stacking (section 4.6.3). The question is: can we combine this technique with stereo photography? Yes, we can—focus-stacked 3D is possible and does, in fact, enhance the 3D impression. First set up your camera (or your cameras) for stereo work. If your camera has a manual focusing mode, switch it on. Then go to **ALT** > **MENU** > *Brackt/Override,* enable the entry *Focus override,* choose an appropriate *Focus step-size,* set the *Focus mode to* **Manual,** and select

5 The same componenent is used in the *Hugin* panorama stitcher (section 6.1).

6 You can easily make 3D glasses yourself. See www.videojug.com/film/how-to-make-3d-glasses.

under *Bracketing type* the number of images that should go into one focus stack. Then return from the *<Alt>* mode and set your camera's shooting mode to *Continuous*. Shoot the required number of images, shift the camera, and shoot the next series.

When composing the image, first perform the focus stacking for the left and the right images separately. The two resulting images are then combined with the *StereoPhotoMaker*.

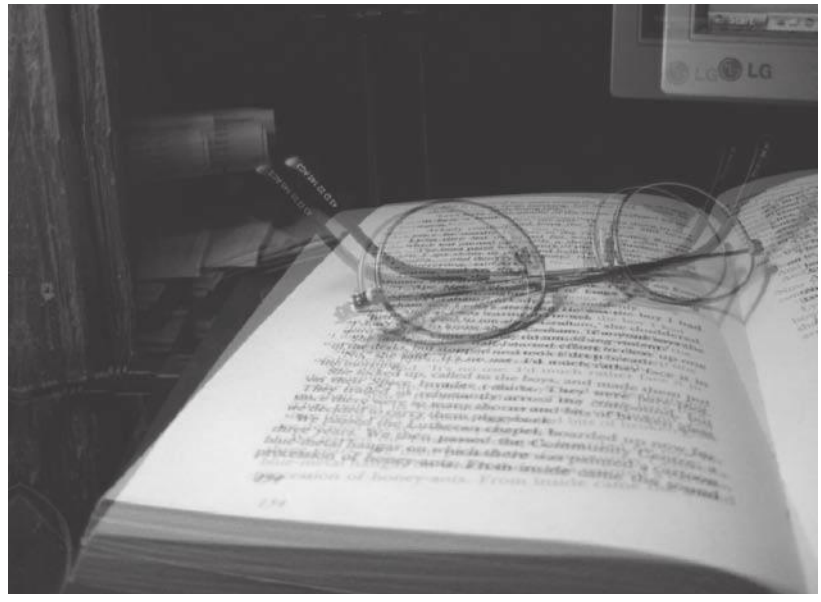


Figure 7-10
This image was composed of 2 x 5 images. Focus stacking was performed with *CombineZP;* the resulting images were then composed into an anaglyph with the help of *StereoPhotoMaker*. The nearest point to the subject matter was approximately 27 cm, and a camera spacing of approximately 15 mm was used, a bit more than what was required. Although the foreground is not completely sharp (some more images would have been needed for a larger depth of field), the resulting 3D impression is very strong when viewed with red/cyan goggles (a color version for viewing is found on the book web site at www.photozora.org/cchm/)«. Canon Digital Elph SD1100 IS, 38 mm, f/2.8, 1/100sec, ISO 400.

7.7.4 Synchronized cameras

After you whet your appetite by taking stereo images with a single camera, you might want to consider acquiring a second camera. Your best option is to buy a camera of the same type (it's always good to have a backup). The advantage is that taking stereo photos with two cameras is much more convenient than with a single camera, and you can take stereo images of moving objects, too. The SDM can synchronize two cameras down to 1/16,000 of a second, so your subjects can move very fast indeed.

Triggering

Synchronization can only be achieved by triggering both cameras at the same millisecond. Special remote controls are required. If you are happy with the simple remote control presented in section 4.9.2, you can easily solder a second mini-USB plug to the first one. This will do the trick.

If you don't want to do any soldering, you can acquire a ready-made unit. For example, *gentStereo* from *Gentles Limited* allows you to control two or more cameras simultaneously. Similarly, the *SDM Canon USB remote* from *digi-dat* can fire up to 12 cameras simultaneously. Or you can use the *Ricoh CA1* remote control with a *mini-USB/mini-USB Y-Cable Charger Adapter*.

In any case, when you want to trigger two cameras synchronously, you must enable the synch mode on both cameras with **ALT** > **MENU** > *Stereo* > *Synchronization* > *Enable synch.* The SDM header line will display "Synch:" followed by the specified delay value (see below). From now on, you should operate the camera only via the remote control. Pressing the shutter button directly will disable the sync mode!

Calibration

Exact synchronization is only achieved when both cameras have exactly the same delay time between USB signal and shutter release. Two cameras of the same model should have approximately the same delay times, but when using different models, differences in delay times can be considerable. Fortunately, you can assign a different delay time to each camera in **ALT** > **MENU** > *Stereo* > *Synchronization* > *User delay.*

Finding the right delay intervals can be tricky. If you still have a CRT monitor (or if you can borrow one or cheaply pick one up at *Ebay*), you can use the *Camera Sync Tester* from www.3dtv.at/Knowhow/Synctest_en.aspx to determine the differences between the delay intervals of two cameras.

If not, you can simultaneously shoot the turntable of a power drill with both cameras. If the drill turns with 3000 rpm, a difference of one degree in the position of the turntable is equivalent to a delay of 1/18,000 sec (0.06 ms). A power hand blender with its higher speed (Figure 4-23) would be even better. By overlaying both pictures, you can determine the difference between the two cameras quite precisely.

Mounting

The simplest solution for joining two cameras is to mount them on a plain bar. The bar can have a simple design, and by default the lens centers are aligned to the same height.

A better mount, however, is the *Z-frame*. The cameras are mounted side-by-side with one camera turned 180°. The advantage of this frame is that—because most cameras are asymmetrically built—the lenses can be mounted closer together, resulting in a lower near limit. Typically, *Z-frames* are built with a special camera model in mind so that the lens centers are aligned.

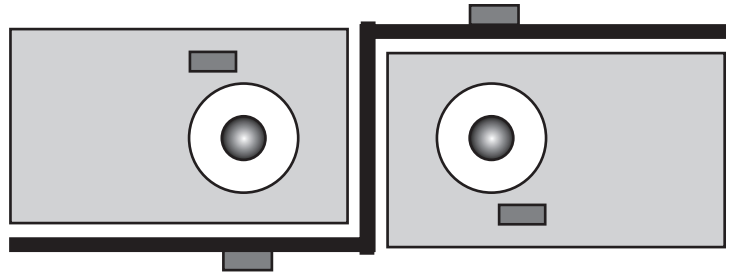


Figure 7-11
The Z-frame can be used to mount two cameras as close together as possible.

If you are skilled, you can build such bars by yourself. You can certainly find instructions on the Internet, for example at http://3dbruce.blogspot.com/. Another option is to acquire a ready-made bar from a manufacturer, for example at www.digi-dat.de/index_eng.html.

Finally, if you want to shoot in portrait orientation, you need a *U-frame* to mount both cameras.

The SDM supports all of these mounts. Under ALT > MENU > *Stereo* > *Cam orientation,* you will find the options **H** (for a horizontal mount), **V** (for a 90-degree mount), and **I** (for a 180-degree mount). Under ALT > MENU > *Stereo* > *Cam rotated,* you will find the options **L** (to the left) and **R** (to the right). For each mode, the camera will present the SDM menus and the SDM OSD with an appropriate rotation (0, 90, 180, 270 degrees). Even the LEFT, RIGHT, UP, and DOWN buttons are swapped accordingly. The native camera menus, however, will remain unchanged.

You also have to identify the position of each camera under ALT > MENU > *Stereo* > *Twin-Cam position* with **L** for left and **R** for right. Some functions will behave differently for left and right cameras.

You can get all of this set up for you by using one of the ready-made configurations (section 7.1) contained in the SDM distribution.

Registering

Registering two cameras is, of course, a bit different than working with a single camera. For example, you will not be able to use the *Edge Overlay* or to view the result as a composite anaglyph on the camera display. The two cameras simply cannot exchange their image data. However, we can rely on grids and on the deviation indicator. The rules used for determining the *Camera Spacing* and the *Deviation Factor* are the same as when working with a single camera:

- Set the camera's *AF mode* to **Center**.
- Under ALT > MENU > *Stereo* > *Camera Spacing,* dial in the distance between the lens centers of both cameras.
- In the same menu, specify a preferred *Deviation factor* (section 7.7.1). The maximum acceptable deviation `dmax` is computed by dividing the image width by the deviation factor.

- By repeatedly pressing **ALT** > **LEFT** or **ALT** > **RIGHT,** browse the SDM OSD until you reach a screen entitled "TWIN CAMS".
- Zoom to the telephoto position (mandatory!). Focus on the nearest object and half-press the shutter button. The entry "Near" in the OSD screen will be updated. It will turn red if the subject distance is too near (section 7.7.1).
- Now focus on the farthest object and half-press the shutter button. The entry "Far" will be updated.
- Now zoom back to the desired zoom position. The calculated deviation (in mm) is displayed in the entry *Deviation*. This value is the maximum deviation required for the current scene (based on the captured distances). It should be lower than `dmax`. A horizontal bar graph shows the *Deviation* value as a percentage of `dmax`. If the bar graph is red, the *Deviation* value is too big (more than 100% of `dmax`). In that case, you should consider zooming out more or moving farther away from the subject.
- Now you are ready to display the guidelines. First, disable the option **ALT** > **MENU** > *Stereo* > *Disable deviation guideline* on both cameras. Also on both cameras, go to **ALT** > **MENU** > *Advanced Menu* > *OSD parameters* > *Grid settings*. Enable the option *Show grid lines* and invoke the function *Load grid from file...* to load `Deviation_H.GRD` (`DEVIAT~1.GRD` on DryOS), align the centerline of the left camera with a significant point on the nearest object. Do the same with the centerline (the left of the two lines shown) of the right camera. Next, align the centerline of the left camera with a significant point on the farthest object and do the same with the **right** line (deviation indicator) of the right camera. The left camera does not show a deviation indicator.

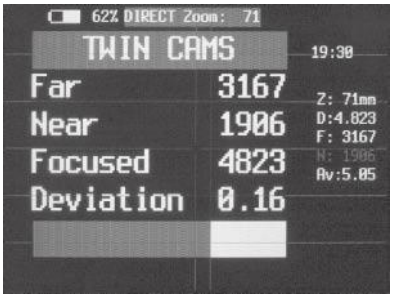


Figure 7-12
The Twin Cams OSD. Below the Deviation entry is the bar graph that is within limits. If it were not within limits, the bar would be red and you would need to zoom out, increase the distance to the subject, or reduce the spacing between the cameras.

7.7.5 Synchronized flash

When working with a twin camera set-up, it is also possible to use the built-in flash units of both cameras. To enforce identical exposure settings on both cameras, it is necessary that both cameras do fire the flash. All you have to do is:

- Set up both cameras as twin cameras as usual. Also enable the option **ALT** > **MENU** > *Stereo* > *Synchronization* > *Add User and flash delays*.
- If you use *Overrides* (section 4.3.1), use the same *Tv* values (shutter speeds) for both cameras. If the cameras are set to automatic exposure, the cameras will set the exposure time to 1/60 sec.
- Turn the flash on (**AUTO** or **ON**) on both cameras. If your cameras support second-curtain mode for flash, be sure to set the flash to *first-curtain mode*.

- When using the *Ricoh CA1* or a compatible remote control, first half-press the button of the remote control to focus. Then press the button fully. The cameras will fire the preflash to determine the exposure. Wait until the blue LED indicates "ready to shoot". Then release the button to fire.
- When using a simple *switch-and-battery* remote control, press its button for less than half a second to focus. Then press the button again to fire the preflash. Release the button after the blue LED indicates "ready to shoot", and the camera will fire.

Only the flash of the right camera is fired with full power. The power of the left flash is reduced by four f-stops in order to avoid double shadows.

7.8 Digiscoping

Digiscoping is the technique used to connect a digital camera with a telescope. It is particularly popular among bird and other wildlife watchers. Compact cameras are well suited for that purpose and allow for an affordable digiscoping solution. The SDM supports digiscoping with various features:

- Even cameras that do not feature a manual focusing mode can be focused on the virtual image projected by the telescope.
- The camera is switched to FAST mode so that it immediately fires the shutter when the switch of the remote control is pressed (you will work with tripod and remote control when digiscoping).
- The camera display is not darkened by the nonactivity timeout (important for wildlife photography).
- Focus bracketing is supported. This can be very useful because the *Depth of Field* (DOF) is very small.

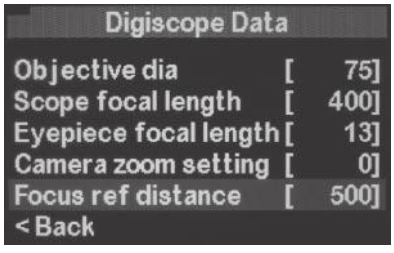


Figure 7-13
The *Digiscope Data* submenu

Before setting up the SDM for digiscoping, set the time-out value for the display to more than one minute. This allows the SDM to stop the camera from timing out at all. Cameras with manual focus mode must be switched to this mode. Then invoke the submenu **ALT** > **MENU** > *Advance Menu* > *Digiscope*. Fill in the specifics of your telescope:

- *Objective dia*. The diameter of the front telescope lens.
- *Scope focal length*. The focal length of the telescope in mm.
- *Eyepiece focal length*. This is the focal length of the telescope divided by the magnification.
- *Camera zoom setting*. The zoom step to be set up at the camera. Widest angle is 0, next step is 1, etc. When starting to shoot, the camera will automatically zoom to this preset value.

- *Focus ref distance*. The distance of the virtual image projected by the telescope lens and the subject distance to which the camera will focus. 500 mm is a good starting point.

After you have entered these values, you can review them in *Record mode* using the DOF browser (ALT+LEFT/RIGHT).

Figure 7-14
The DOF browser showing the *Digiscope Data*. The entry *Camera EP* is colored red, indicating that the camera's entrance pupil *(Camera EP)* is the limiting aperture. In the ideal case, it will match the *Scope EP*.

- *Scope mag*. The magnification of the scope.
- *Scope EP*. Diameter of the scope's exit pupil in mm. This entry will be red when it is the limiting aperture (smaller than the camera's entrance pupil).
- *Camera EP*. The diameter of the camera's entrance pupil in mm. This entry will be red when it is the limiting aperture (smaller than the scope's exit pupil).
- *EFL*. The combined focal length of camera and scope.
- *EFL (35mm)*. The 35mm-equivalent of EFL.

It's a good idea to use focus bracketing in connection with digiscoping. First, it increases the tolerance for focusing. Also, the series of images obtained can be combined via focus stacking (section 4.6.3) in order to increase the depth of field. With digiscoping, the DOF is notoriously small, so focus stacking can make a lot of sense.

- In the *Brackt/Override* submenu, set *Focus mode* to **Digiscop**. This will cause bracketing around the preset *Focus ref distance*. The step size will also depend on this value: the lower the *Focus ref distance,* the smaller the focus step size.
- Set *Tv bracket value* to **Off**.
- Choose any value from 3 times to 19 times in entry *Bracketing type*.
- Exit *<Alt>* mode and set the camera to *Continuous mode* or to *Custom timer* (with an appropriate number of shots).

Of course, the scope must be mounted on a sturdy tripod. The camera must be mounted on the same tripod, so you need some kind of a bracket (digiscoping adapter) on which both scope and camera can be mounted and adjusted. The ideal adapter would allow swinging the camera quickly out of place (for access to the ocular) and swinging it back again without compromising stability. Such adapters are not cheap—they may even cost more than your camera. For the occasional shot, however, this low-cost and low-tech version may be an alternative:
www.videojug.com/webvideo/how-tomake-a-digiscoping-adapter.

You also need to trigger the camera with the help of a USB remote control. Because the **Digiscop** focusing mode switches the camera to FAST mode, the camera will fire immediately when you press the switch instead of waiting for switch release. If you don't have a remote cable, you could

use the *Custom timer* with a few seconds delay to fire the camera (obviously not an option for action shots). Anything else will lead to shattered or unsharp images.

After setting up your tripod and camera, you should carefully focus the camera/scope combination. This is done with the help of the **scope's** focus control while looking at the camera display. For better control, you may want to switch the camera's screen magnifier on. In the native camera menu, set the *AF Frame* to **Center** and the *AF-Point Zoom* to **On**. Upon half-pressing the shutter button, the center AF frame is enlarged—but only if this area is sharp enough. So the *AF-Point Zoom* can act as an in-focus indicator.

7.9 Scripting

Not all *uBasic* commands and not all CHDK commands are supported by the SDM. For example, the SDM does not know the *uBasic* instruction `select` (section 5.3.5) or the CHDK command `get_display_mode` (section 5.5.8).

On the other hand, the SDM introduces a large number of new commands. Some of them are synonyms of existing CHDK commands. Their only rationale is to make the script more readable. For the same reason, it is also possible to omit the `print` command and simply write a string into a script line. For example:

```
"Hello SDM!"
```

will output

```
Hello SDM!
```

on the camera's display. Other commands simplify tasks that would need elaborated scripts in the CHDK, especially in the areas of bracketing, time lapsing, and communications. And, the SDM can do one thing that is not (yet) possible in the CHDK at all: switching between photo and video mode.

The following is a small sample script that waits on a signal from the USB port (from a remote control, an RC wireless, a microcontroller, etc.) and then shoots a movie for the specified number of seconds:

```
@title Movie
@param t Duration(sec)
@default t 5
"Waiting on Remote"
wait_for_switch_press
"Shooting movie"
"Please stand by"
shoot_movie_for t
end
```

The SDM has also found a way around the slow speed of *uBasic* scripts. In the CHDK, each script line (except comment lines) is executed in a new time unit (10 msec). With the SDM, you can control the speed of execution. The command:

```
set_script_speed s
```

accepts a value between 1 and 5, determining the number of script lines to be executed within one time unit if physically possible. This setting can be retrieved with:

```
get_script_speed s
```

SDM scripting can be quite different from CHDK scripting, and many CHDK scripts will not run under the SDM without modifications. Fortunately, there is an SDM version of the UBDB debugger (section 5.8), the *SDMUBDB* that can be downloaded from www.zenoshrdlu.com/kapstuff/zsdmubdb.html. So you can do the testing and the script conversion conveniently on a PC.

Button-related commands

`take_photo_now`
Identical to `shoot` (section 5.5.1).

`shoot_movie_for s`
Switches to *Movie mode,* records a movie for the specified time (seconds), and then returns to *Record mode*.

`wait_for_switch_press`
Waits until a signal is present on the USB V+ pin (section 4.9.2).

Exposure-related commands

`nd_filter_off`
`nd_filter_in`
`nd_filter_out`
Identical to:
`set_nd_filter 0`, `set_nd_filter 1`, `set_nd_filter 2` (section 5.5.2).

Focus-related commands

get_focused_distance d

Identical to `get_focus` (section 5.5.3).

set_focus_to d

Identical to `set_focus` (section 5.5.3).

get_focus_ref p

set_digi_focus_ref_to

Retrieves and sets the focus position used by the digiscoping function (section 7.8).

lock_autofocus

unlock_autofocus

Identical to `set_aflock 1` and `set_aflock 0` (section 5.5.3).

Zoom-related commands

set_zoom_to_step z

set_zoom_to z

Identical to `set_zoom` (section 5.5.4).

Bracketing-related commands

save_stack

Creates a log file in folder `CHDK/STACKS` recording the focus or *Tv* values of a bracketing series. It should precede any other bracketing command.

each_photo_alternating

each_photo_darker

each_photo_lighter

Used before an exposure *(Tv)* bracketing sequence is started. This only works with the *Custom Timer*. The *Continuous Shooting mode* always alternates.

hdr_bracket_1/3_ev_steps s

Used to set the step width for exposure *(Tv)* bracketing in 1/3 EV (f-stops).

auto_focus_bracketing

Sets up the camera for a focus bracketing sequence to take a sequence of images starting at a defined subject distance and ending at infinity. The steps increase with distance. Cameras with manual focus must be in manual focusing mode. Also, the *Custom Timer* or the *Continuous Shooting mode* must be active.

equal_step_focus_bracketing

Sets up the camera for a focus bracketing sequence, but with fixed focus steps beginning at a start subject distance. This is typically used for focus stacking in macro mode. Cameras with manual focus must be in manual focus mode. Also, the *Custom Timer* or the *Continuous Shooting mode* must be active.

set_focus_step_to

Sets the size of the focus step used by the command `equal_step_focus_bracketing`.

digiscope_bracketing

Enters the bracketing mode for *Digiscope* mode (section 7.8). The camera will focus before and behind the virtual image of the scope. Cameras with manual focus must be in manual focusing mode. Also, the *Custom Timer* or the *Continuous Shooting mode* must be active.

number_of_images_to_capture_is n

Specifies the number of images in bracketing mode.

start_continuous_sequence

end_continuous_sequence

Starts and ends a bracketing series when the camera is in *Continuous mode*.

start_custom_timer_sequence

Starts a *Tv* or focus bracketing series when the camera is in *Custom Timer mode*. The series will end after a given number of images have been taken. The number of images is specified in the *Custom Timer* under *Shots*.

bracketing_done r

Returns 1 when the bracketing series is complete.

wait_until_done

Waits until a series of photos taken with the *Custom Timer* or under the *Continuous mode* has been completed.

Time-related commands

sleep_for t

sleep_for_msecs t

Identical to `sleep` (section 5.5.7).

sleep_for_seconds s

Equivalent to `sleep s*1000`

sleep_for_minutes m

Equivalent to `sleep m*60000`

end_time h,m

finish_time h,m

Specifies a time of day when an operation should end. Must be specified before `sleep_until` or `start_time`. Currently, only the command `time_lapse` supports this feature.

sleep_until h,m

start_time h,m

The camera waits until the specified time of day (`h` = hours, `m` = minutes). During that time the screen is darkened.

time_lapse a, b, c, d, e, f, g, h, i, j, k, l, m, n, p, q, r, s, t, u

Starts a time-lapse series. This command is the core of the time-lapse script contained in the SDM distribution. Here, it can be called as a command from another script.

Parameters:

- `a` First delay minutes
- `b` First delay seconds
- `c` Shoot interval minutes
- `d` Shoot interval seconds
- `e` Number of repeats
- `f` 1 = endless mode (`e` ignored), 0 = limited by parameter e
- `g` 0 = single, 1 = continuous, 2 = custom timer, 3 = burst
- `h` Number of exposures for *Tv* bracketing
- `i` Number of exposures for focus bracketing
- `j` 0 = *Tv* bracketing, 1 = focus bracketing, 2 = both
- `k` 0 = lighten, 1 = darken, 2 = alternate
- `l` *Tv* bracketing step in 1/3 EV
- `m` 0 = equal step focus bracketing, 1 = autofocus bracketing, 2 = Digiscope bracketing
- `n` Focus step in mm.
- `p` Digiscope focus distance
- `q` 0 = don't blank screen, 1 = blank screen
- `r` 0 = don't shutdown, 1 = automatic shutdown after required number of pictures, 2 = shutdown by USB signal
- `s` 0 = don't save bracketing stack, 1 = save bracketing stack
- `t` 0 = no sunrise, 1 = sunrise mode. The *Sunrise* menu controls exposure and number of images when set to 1.
- `u` 0 = use TXT format for sunrise log, 1 = use CSV format for sunrise log

Data transmission-related commands

enable_usb_download

disable_usb_download

By default, USB download (upload) is disabled, so that the camera can react to a USB remote control. By enabling the USB upload and switching to *Playback mode*, a peer program running on a PC (such as the WIA loader) will start and open a connection.

make_usb_connection

Switches to *Playback mode* and opens a USB connection for upload.

break_usb_connection

Closes the USB connection and switches back to *Record mode*. The peer program on the PC must have already closed the connection.

upload_images_for t

Switches to *Playback mode* and opens a USB connection for the specified time. Then it closes the USB connection and returns to *Record mode*. Can be used in combination with the WIA loader (section 7.6) for uploading images at regular intervals.

unit_pulse

Generates a single pulse on the PRINT LED. Typically used to calibrate the connected external device (section 7.6).

send_data a, b, c

Sends three bytes of data to an external device by flashing the PRINT LED (section 7.6). The external device acknowledges the correct reception by sending a signal to the USB V+ pin. The values `a`, `b`, and `c` should be in the range of 0.255.

data_received a

Used after `send_data` to wait for an acknowledgement by the external device on the USB port. Waits for 100 msec and returns 1 if a signal is received at the USB V+ pin within that time. Returns 0 otherwise.

LED and sound-related commands

af_led_off

af_led_on

Identical to `set_led 9, 0` and `set_led 9, 1`. (section 5.5.8). Be careful with the `af_led_on` command—the AF LED is probably not built to be illuminated for a long period of time.

blink_af_led_for t

Blinks the AF LED for the specified time `t` (seconds). Typically used for kite aerial photography (KAP) because the AF LED can be seen from some distance.

blue_led_off

blue_led_on

Identical to `set_led 8, 0` and `set_led 8, 1` (section 5.5.8).

beep

Identical to `playsound 4` (section 5.5.8).

Camera state-related commands

lcd_on_off

Toggles the on/off state of the camera display. This does not work on all cameras.

turn_backlight_off

turn_backlight_on

Identical to `set_backlight 0` and `set_backlight 1` (section 5.5.8). Works on all cameras.

get_shooting_mode r

Identical to `get_shooting` (section 5.5.10).

movie_mode

Switches the camera to movie mode (except on cameras with a separate **MOVIE** button).

playback_mode

Switches the camera to *Playback mode*.

record_mode

Switches the camera to *Record mode*.

get_sync a b c d

set_sync a b c d

Retrieves and sets the SDM synchronization state: Parameter a goes into *Enable Synch* (1 = enable, 0 = disable), parameter b goes into *Enable Synch delay* (1 = enable, 0 = disable), parameter c goes into *User delay* (in 0.1 msec), parameter d goes into *Coarse delay* (in 0.1 sec) (section 4.9.3).

sync_on

sync_off

Enables/disables the *Enable Synch* option (see `set_sync` above).

8 Kites, Balloons, and Multikopters

Despite its name, the *Stereo Data Maker* is probably most often used for *Kite Aerial Photography* (KAP) and other kinds of remote aerial photography. In particular, its excellent support for remote operation, time lapsing, and the integration of external devices via a serial interface has gained the SDM a good reputation in the KAP community.

The original version of the CHDK is also used often for remote and unattended operation. In particular, its support for the *Lua* scripting language makes it suitable for very sophisticated applications. For example, one of the weather balloon missions mentioned in the introduction of this book used a slightly modified version of the original CHDK. A *Lua* script handled the scheduling of both photo and video operations, as well as a shutter-priority exposure control.

8.1 Kite Aerial Photography

In *Kite Aerial Photography* (KAP), a camera is lifted by a kite and takes photographs autonomously or when triggered by a remote control. Small, lightweight cameras such as the *Canon Digital Elph SD (Ixus)* series are ideal for KAP. Nearly any stable single-line kite design can be used to lift such a lightweight camera. An interesting option for KAP—although expensive—is the *Helikite* from *Allsopp Helikites Ltd.,* a combination of a helium-filled balloon and a kite. The advantage of this design is that it can be operated in both windless and very windy conditions, so you don't have to rely too much on the weather—quite important for the professional photographer.

Typically, the camera is mounted on a rig which is suspended somewhat below the kite. By means of gravity, the rig and the camera are kept aligned with the horizon. There is still some movement, but less than there would be with a camera directly mounted to a kite. Nevertheless, short shutter speeds are recommended, and the camera's *Image Stabilizer* (IS) can also help.

In its simplest form, the camera is mounted directly on the rig in a fixed position and performs the operations autonomously controlled by a time-lapse script. The script should allow for an initial delay, so that the kite can reach its working height before operation starts.

Alternatively, the camera can be controlled from the ground. A very simple (but sometimes awkward) solution is a thin pair of wires connected to the USB port of the camera (section 4.9). A more comfortable solution is a radio-controlled (RC) system. One solution is to let the RC system trigger the camera mechanically (via a servo operating the shutter button). A more elegant solution is to replace the servo with a device converting the RC signal into a USB pulse. Devices such as the *gentled chdk2* even allow (by means of two RC channels) operating up to six configurable camera functions (e.g., shutter, zoom, exposure, focus, etc.). Advanced RC systems such as the *DuneCam* from *Dunehaven Systems* can even relay the display image back to the ground unit, thus giving the photographer some feedback.

A more advanced method to mount the camera is a rig that can rotate the camera vertically and horizontally with the help of servos. Some can even rotate the camera along its optical axis and can thus change the camera's orientation from landscape to portrait. The servos are usually controlled via the RC system but can also be controlled via the camera's serial interface (blue LED) as provided by the SDM (section 7.6). Another option is using an external controller such as the *Automatic Rig Controller* (AuRiCo) that can control both the rig and a CHDK-enabled camera through the USB port.

Especially when shooting video, the motion of the rig can become annoying to the viewer. Most of this motion can be cancelled out by using *Gyro Servos* (again from *Dunehaven*) instead of normal servos to control the rig. Small on-board gyroscopes measure the movements of the rig and advise the servos to counteract those movements.

8.2 Balloon-based photography

While KAP is dominated by dedicated amateurs, professional photographers often turn to balloons as camera platforms. A balloon is not as dependent on the weather and is easier to control. When carrying a large professional camera, the balloon, of course, needs to be quite large—and the helium will be expensive, too. But with a small digital compact camera, the demands on the balloon are minor and balloon photography can be done at an affordable scale. Even a stack of cheap party balloons could do the job. Nevertheless, transport and storage of balloons remains a challenge if you don't want to refill the helium for each mission.

Except for the balloon, all of the other required components (such as rigs and RC systems) are the same as for KAP photography.

A different category is balloon-based photography in high altitudes, as mentioned in the introduction to this book. In this case, there are additional requirements: the equipment must be protected against cold temperatures, and you must provide a way to safely retrieve the camera, such as a parachute and a tracking device.

8.3 Motorized flying platforms

Other flying platforms are usually motorized and include RC-controlled model airplanes and helicopters. For photography, RC-controlled paragliders are an interesting option. They fly more slowly and give you more time to take pictures. Normal aircraft, in contrast, require lots of attention during flight. Taking pictures at the same time can be quite demanding—it can be a job for two people, one acting as pilot, the other as photographer.

Multikopters are in a different league altogether; they are floating airbound working platforms for all kinds of tasks. Examples are the *MikroKopter* DIY project (www.mikrokopter.de/ucwiki/en/MikroKopter) and its commercial cousin, the *AirRobot*. Multikopters can operate autonomously or be RC-controlled. Each device consists of an array of vertically oriented propellers (3–8) powered by lithium-polymer accumulators. Sensors and a control unit keep the craft stable in the air; the device can even autonomously follow a predefined path guided by GPS. Once positioned over an object or a scene, the photographer can concentrate on taking photos—the Multikopter takes care of itself.

With all motorized crafts, there is the problem of vibration. It is necessary to choose a short shutter speed and to use shutter priority exposure control. On small diaphragm-less cameras such as the *Canon Digital Elph (Ixus)* that are not equipped with such a mode, this can be achieved by using a script (section 5.7.4). Also, it will probably be necessary to switch the *Image Stabilizer* (IS) off because it usually worsens the problem. In any case, tests should be made to find out the best shutter speed and whether the image stabilizer can remain active or not.

8.4 Other unattended operations

The remote operation of a camera is not limited to kites and other airborne vehicles. Any kind of remote-controlled vehicle can be used as a camera platform—on land, sea, in the air, or in space. Often such vehicles are used to take pictures in areas where humans can't go or areas that are too dangerous.

Unattended operation is not only useful for mobile platforms but also for stationary use. Surveillance is one application area. The camera is installed in a certain position and takes pictures at regular intervals or when triggered by motion. Multishot techniques are also candidates for automated shooting sequences. The creation of large panoramas such as *Gigapixel* images can and should be performed autonomously by the camera in combination with a robotic panorama head. The creation of timelapse videos is another candidate for unattended operation. Shooting hundreds of pictures manually is simply too boring and too error-prone.

When such missions last for a long time, it is necessary to evaluate the power supply and provide the camera with external power, e.g., from a vehicle's batteries. A voltage regulator is needed to match the supply voltage with the voltage required by the camera. Also needed is a way to feed the current into the camera. A simple solution is salvaging a cheap replacement battery and connecting its contacts with the output of the voltage regulator. Alternatively, a dummy battery pack can be made from cardboard using paperclips as contacts. In a similar way, it is possible to supply the camera with current from the USB port of a notebook computer or from a solar panel. Most stationary-mounted cameras, however, can be supplied with current from an AC outlet by using the standard AC power supply unit from the camera manufacturer.

If the camera is not accessible, it is worthwhile to install a backup battery, too. If the power shuts down due to a supply current dropout, there is no way to turn the camera back on again—except, perhaps, via a robotic arm that presses the power button. Also, before a mission, make absolutely sure the *Auto Power Down* function of your camera is switched **off**.

If the camera is running on its own battery, power saving is essential. In most cases, the camera display is not used during the mission, so the display backlight can be switched off. If the script operating the camera does not switch the backlight off, you can insert a spare AV plug into the camera's AV OUT terminal, which will switch off the display backlight.

Another problem is that after a shorter or longer period of time, the memory card will be full. Exchanging the memory card can sometimes be cumbersome. Basically, there are two options to solve the problem. If the camera can be connected to the USB port of a nearby PC, the images can be downloaded to the hard disk of the PC by using the SDM in connection with the WIA loader (section 7.6). The other option requires a suitable Wi-Fi hotspot, a Wi-Fi router, or a Wi-Fi enabled computer nearby. If this is available, the camera's memory card can be exchanged against an *Eye-fi* card. The Pro version of this card can hook into different wireless networks and upload images to a photo-sharing site on the Web or to a computer.

9 A Look across the Fence

The success of the CHDK seems to have a ripple effect. Owners of other cameras start to ask: why can't I get something like that for my camera? In fact, quite a few similar projects exist by now—projects that are targeted at cameras other than the compact models from Canon. Some are still in their infancy, but others already have deliverables.

9.1 Canon EOS CHDK

Currently, the CHDK only works on *Canon Powershot* and *Digital Elph (Ixus)* cameras. However, some people have started to port the CHDK to Canon EOS cameras such as the *EOS 40D*. So stay tuned. Information is available at http://chdk.wikia.com/wiki/Category:DSLR.

9.2 Canon 5D as a professional movie camera

Using DSLRs as professional movie cameras is not a bad idea at all. These cameras are able to deliver video in HDTV quality with up to 24 frames per second and more. They have interchangeable lenses, and the standard focal length is similar to that of traditional movie cameras—so you get that classic movie look with its rich tonal range and well-controlled depth of field. For example, close-up shots of actors with a very unsharp and quiet background *(Bokeh)* are typical. Last, but not least, DSLRs are a LOT cheaper than professional movie cameras.

One of the cameras suitable for movie work is the Canon 5D Mark II, a full frame (35mm) DSLR. This camera is subject to the project *Magic Lantern* (hosted at http://magiclantern.wikia.com). The project aims to enhance the 5D with some missing features required for serious movie work, such as:

- On-screen audiometers
- Manual gain control for audio with no AGC, resulting in less audio noise and less hiss
- Zebra mode for exposure control
- Crop marks for 16:9, 2.35:1, 4:3 and any other format

- Control of focus and bracketing in video mode
- Scripting using the *PyMite* (a Python subset for microcomputers) scripting language

The software is readily available for the 5D, and a new version for the 7D has been announced.

9.3 Pentax hacks

The *Pentax Hack* project (http://pentax-hack.info) is still in the early stages of analyzing the camera's native firmware. The project is targeted at the Pentax K10D/K20D and Samsung GX10/GX20 cameras.

Appendix

A.1 Using cards with more than 4 GB capacity

Using very large memory cards can sometimes be necessary when doing video or time-lapse work where the camera runs unattended, and it is inappropriate to change the card frequently.

The simplest option is to launch the CHDK manually after each camera start with the function *Firm update...* as outlined in section 3.4. This should work for most cameras.

The trouble begins, however, if you want to use the AUTORUN feature. Partitions with more than 4 GB size MUST be formatted with FAT32, and the CHDK does not boot from FAT32. The trick here is to run two partitions. The first partition is formatted with FAT16 and is quite small: 2 MB is sufficient. This partition only contains two files: `DISKBOOT.BIN` and `PS.FI2`. All other files and folders are stored on the second partition that occupies the rest of the memory card and is formatted with FAT32.

Unfortunately, only a few cameras support multipartitioned cards. For supported cameras, you will find the two entries *Create card with two partitions* and *Swap partitions* under **ALT** > **MENU** > *Miscellaneous Stuff*. With these menu functions, you can format a large card with two partitions[1]:

1. Install the CHDK on the card in the usual way. Load the CHDK with the function *Firm update...* as outlined in section 3.4 and make a backup of your card.
2. Now invoke the function **ALT** > **MENU** > *Miscellaneous Stuff* > *Create card with two partitions*. This will create a FAT16-formatted partition of 2 MB.
3. Copy the files `DISKBOOT.BIN` and `PS.FI2` to this small partition.
4. Load the CHDK again using the function *Firm update...*
5. Invoke **ALT** > **MENU** > *Miscellaneous Stuff* > *Swap partitions* to hide the first partition and make the second partition visible.
6. Now format the card again. This will only format the second, large partition. Because this partition is larger than 2 GB, it will be formatted with FAT32.

1 The process reminds me a bit of the wolf-goat-cabbage-riddle: You have to bring all three safely to the other shore, but your boat will only take one of them.

7. Copy the CHDK files `DISKBOOT.BIN` and `PS.FI2` onto this partition, too, so that later we can also boot from this partition. By now, we should have the CHDK on both partitions.
8. Load the CHDK again using the function *Firm update*...
9. Now perform **ALT** > **MENU** > *Miscellaneous Stuff* > *Swap partitions* again to make the first, small partition visible.
10. Invoke the function **ALT** > **MENU** > *Main Menu* > *Miscellaneous stuff* > *Make card bootable* to enable the AUTORUN function. This will make the first partition bootable.
11. Write-protect the card and switch the camera on again. It should now perform an AUTOSTART from the first, small partition. When the CHDK detects a second partition formatted with FAT32, it will switch all access functions to the second partition. Until the next start, the first, small partition will not be used again.

There is one more problem when you want to exchange files with a computer running under Windows. Windows can only see the first partition. To make your files accessible to Windows, you must first swap the partitions via **ALT** > **MENU** > *Miscellaneous Stuff* > *Swap partitions*. Afterwards, you must swap partitions again—to do so, you may need to boot from the FAT32 partition using the function *Firm update*....

A.2 Troubleshooting

Problem: After invoking the CHDK menu or one of its submenus, the menu sometimes vanishes after a few seconds.
Cause: The CHDK menu is overwritten when the native firmware updates the information on the screen. This happens, for example, shortly after start-up when the display is in the "no information" mode. It also happens in *Replay Mode* when you change the orientation of the camera, or when you switch camera modes. In all of these cases, the native firmware repaints the screen and the CHDK menu is destroyed.
Workaround: All buttons remain operational. Simply press the **UP** or **DOWN** button and the CHDK menu should reappear.

Problem: When running a script, the message "assert failed—game over!" appears and the camera shuts down.
Cause: Some script commands have brought the camera into an illegal state. The camera shuts down in order not to damage any hardware.
Solution: Check the script. Consider using simulated key presses instead of changing the camera state directly with commands such as `set_zoom`.

Problem: Script execution fails with an "Out of memory" message.
Cause: One possible cause is that the *Edge Overlay* feature is enabled and has allocated memory for the overlay image.
Solution: Go to the submenu *Edge overlay* and disable the *Edge Overlay* feature. Also, invoke the menu function *Free internal Memory* to release the allocated memory.

Problem: The camera shuts down every time you switch it on.
Cause 1: You called functions from the *Debug* menu in an improper way.
Solution 1: Restore the file `CHDK/CCHDK.CFG` from a backup or delete it. You will lose all configuration changes applied past the last backup.
Cause 2: You enabled the *Autostart* option in the *Script* menu. The current script is faulty and causes the shutdown.
Solution 2: Edit the configuration file `CHDK/CCHDK.CFG` with the configuration editor `CFGEDIT.jar` (see section 4.12). Then reset the *Autostart* option.

Problem: The camera hangs and does not react to key presses.
Cause 1: Possibly a bug in the CHDK.
Solution 1: Remove the batteries, wait for 10 seconds, then insert them again and restart. If the problem persists, reinstall the CHDK. Consider using a more recent version of the CHDK.
Cause 2: You are using a wrong CHDK version.
Solution 2: Make sure to use the correct version matching both camera model and the camera's firmware version (section 3.2).

Problem: You have found a bug in the CHDK.
Solution: First check to see if the bug persists in the newest version of the CHDK. If yes, and if you have access to another camera, find out if the bug is specific to a certain camera model or not. Then file the bug in the *Mantis* bug tracking system (http://chdk.kernreaktor.org/mantis). Before you do, please check to see whether a similar bug is already filed there.

A.3 Web links

Official CHDK website: http://chdk.wikia.com From here you can download the different CHDK builds and get access to all kinds of tutorials, scripts, and other resources.

CHDK Forum: http://chdk.setepontos.com/ Discussion forum for CHDK-related themes.

Omgili Aperture Chdk: http://omgili.com/aperture-chdk Another CHDK-related discussion forum.

Camera Features: http://chdk.wikia.com/wiki/CameraFeatures This page lists findings about the different cameras supported by the CHDK, such as longest exposure time, fastest shutter speed, highest aperture value, lowest and highest ISO, fastest motion detection response, and more.

Motion detector speed test: http://dataghost.com/chdk/md_meter.html Site for testing the speed of motion detectors.

Muttyan's home page: http://stereo.jpn.org/eng/index.html Site dedicated to stereo photography, featuring many stereo-related software products including the *StereoPhotoMaker* and the *StereoDataMaker* (SDM).

This is IT!: http://home.hccnet.nl/s.vd.palen/index.html Home of the *PhotoLapse* time-lapse video composer.

KAP and CHDK/SDM Java Utilities: http://www.zenoshrdlu.com/kapstuff/zchdkstuff.html Site featuring the *UUDB* und *SDMUUDB* debuggers for *uBasic* as well as the *CHDK configuration editor*.

CHDK for balloon photography: http://www.francescobonomi.it/ballon-photography-CHDK Documentation about a balloon-based space mission with a CHDK-enabled camera.

MikroKopter: http://www.mikrokopter.de/ucwiki/en/MikroKopter Official site for the MikroKopter flying camera platform.

dng4ps: http://code.google.com/p/dng4ps2/ Home of the *DNG4PS (DNG for Powershot)* converter.

Magic Lantern Firmware Wiki: http://magiclantern.wikia.com Site for Canon 5D conversion (section 9.2).

Pentax Hack: http://pentax-hack.info/index.html Official site for the enhancement of Pentax DSLRs.

Hack a Day: http://hackaday.com/category/digital-cameras-hacks/ All kinds of (hardware) camera and equipment hacks.

A.4 Contributing to the CHDK

The CHDK is a large community effort, and there are many ways of contributing to this project. It doesn't have to be core development—there are many other areas where you can contribute. People who adapt the CHDK to a new camera are always needed. Each year, Canon comes out with a bunch of new models that are not yet supported by the CHDK.

Besides the camera itself, the first thing required for porting the CHDK to a new camera is a dump of the native camera firmware. This may be easy to obtain by using *CardTricks* (section 3.2) for dumping directly from the camera. For many cameras, however, it may be necessary to use a hardware-based method and dump the firmware through the blue LED. Detailed instructions for different firmware dumping methods are found at http://chdk.wikia.com/wiki/Firmware_Dumping.

In the next steps, the firmware is analyzed. The hooks where the CHDK can lock in must be found. Some of these hooks must be searched for manually by inspecting the code. Others can be found with the help of a tool, *Signature finder* (`finsig`/`gensig`). The mapping of keyboard functions to buttons must be adapted for the specific camera model, too.

You should have basic knowledge of the C programming language and the ARM assembler language if you want to take on such a task. Because all CHDK functions must be tested on the new camera, some time and effort are required. For more information, please see http://chdk.wikia.com/wiki/Porting_the_CHDK.

There are other areas, too, where contributions are welcome. Improving the documentation, writing tutorials and articles, implementing tools and utilities, and last but not least, creating novel and useful scripts, are all valuable contributions. Even filing a bug (appendix A.2) helps improve the software.

A.5 Bibliography

[Lua51Ref] R. Ierusalimschy, L. H. de Figueiredo, W. Celes; Lua 5.1 Reference Manual; http://www.lua.org/manual/

[UserGuide] http://chdk.wikia.com/wiki/File:CHDK_UserGuide_April_2009_A4.pdf

[Gulbins2009] Juergen Gulbins / Rainer Gulbins; Photographic Multishot Techniques; Rocky Nook; 2009

[Howard2008] Jack Howard; Practical HDRI; Rocky Nook; 2008

[Bloch2007] Christian; The HDRI Handbook; Rocky Nook; 2007

Index

Back cover image data

SUBJECT Hamburg Docklands
CAMERA Canon SD1100 IS
ISO 6121
SHUTTER SPEED 1/19
APERTURE f/2.8
FOCAL LENGTH 6.2 mm (equiv 38 mm)

Shot handheld with six single exposures as DNG images.
Superimposed with PhotoAcute.

Berthold Daum

Book Website www.photozora.org/cchm

- Images
- Errata
- Updated software and scripts