LazyConv Image Converter User Guide

version 1.2

Table of Contents

What is LazyConv.

LazyConv integrates existing raw photo conversion and sorting engines in a way that minimizes human time required to process huge amounts of raw photos.

Target user of LazyConv - volume shooter who still doesn't want to compromise quality by relying on in-camera JPEGs. LazyConv offers a trade-off between in-camera JPEG and time-consuming traditional RAW-based flow that should meet the demands of a quality-concerned user.

LazyConv enables you to convert thousands of raw photos into JPEG, TIFF or other format, while

- letting you ***conveniently*** choose a color correction per an image
  - from a limited but user-configurable set
  - virtually any color-correction tool is available for defining the corrections
- minimizing the time you have to interact with the computer
  - all longer-than-instant operations performed in batches off-line

    - pictures requiring special treatment could be conveniently selected

The following software is utilized by the current version of LazyConv:

- Irfanview (www.irfanview.com) – used for raw conversion, as well as for viewing and sorting images
- dcraw (cybercom.net/~dcoffin/dcraw/) – used for raw conversion
- ImageMagick (www.imagemagick.org) – used for basic image manipulations and color corrections
- TCL programming language (see www.tcl.tk) used for two purposes:
    - the whole integration is written in it
    - TCL command shell serves as the main interface with the user – commands for LazyConv are printed at its command prompt
- FreeWrap (freewrap.sourceforge.net) is used as a stand-alone TCL language interpreter - this simplifies LazyConv installation on the user side
- tcllib (tcllib.sourceforge.net) – standard TCL extension library
- TkCon (tkcon.sourceforge.net) - user-friendly TCL console (optional)
- EXIV2 (www.exiv2.org) – (optional) used for reading EXIF fields in the images

Though don't worry, it's much simpler in the field, one doesn't need to be familiar with all these tools to be able of using LazyConv.

To assure that LazyConv is indeed easy to use:

1. There are only 7 (seven) mandatory commands in LazyConv.
2. Most of the time you work with Irfanview as a front end.
3. You need to remember only 8 key bindings in Irfanview: next-image (Space), previous-image (BackSpace), delete-image (Del), copy-image (F8), move-image (F7), go-full-screen (Enter), rotate-JPEG (Shift-J), quit (Esc).
4. When installing LazyConv, you need to change up to 10 directory paths in 3 text files.

The engine-integration nature of LazyConv implies that other raw-conversion and image-processing engines could be incorporated into it in the future; they are only required to provide command-line interface.

Manifesto. Why another converter?

I came to the decision of writing my own photo-conversion tool after bringing ~1200 camera-raw images from a vacation. I happened to have a laptop “in the field”, and from time to time I made sample JPEG-s from all recent images (by Irfanview in batch mode) and sorted out the bad ones (deleted appropriate JPEG-s). I believed this will save me time when I come home.

This process left me with 1200 source (raw) files, 800 JPEG-s chosen to be kept, and a big question how to proceed from this point.

As the output, I wanted to obtain TIFF images corresponding to my 800 “good” JPEG-s, possibly some of them with non-default color settings.

> *Reminder: the **traditional way of raw photo processing** includes per each image one-by-one:*
>
> *(a)wait until a small preview with default settings is made*
>
> *(b)wait a lot more if you want to magnify the preview*
>
> *(c)play with one or more correction tools and see how they affect the preview*
>
> *(d)commit the chosen corrections by either initiating the immediate conversion (and wait till it finishes), or by inserting the image and correction spec into the processing queue*
>
> *Most stages include waiting for some image processing to complete, the delays range from terrible with Dimage Master to small but sensible on Raw-Shooter-Essentials. Specifically, any operations that include raw conversion are permitted to be extremely slow (so LazyConv performs them “offline”). Then, operating standard color-correction tools, while simple and intuitive, is a killer ergonomically when done too many times.*

I didn't see how to do with the traditional approach. Here is the list of problems I had to overcome to make the workflow more suitable for my needs:

1. How to utilize my premature image sorting – I had 400 “bad” raw images that I didn't want to deal with. But the criterion to know the particular raw image is “bad”, was that there was no JPEG made from it. So, the first task was to separate between raw files for which JPEG existed, and those for which JPEG has been deleted. I knew no “common” tool to achieve this, so the need to write some code was clear. Further, those JPEGs are called *previews* of the raw images. Bottom line, the whole LazyConv is built around a loop that goes over all existing previews, finds input (raw) file for each, and performs some kind of image conversion.
2. How to guarantee that the quality of default-color conversion will suffice for absolutely most of the images. I won't agree to manually correct, say, 200 images. The answer was to choose a proper batch converter. I took a set of 12 representative raw images, and converted them with: Irfanview, MRWFormat, Vuescan, PolyView, Raw-Shooters-Essentials. Believe it or not, Irfanview came out a total winner. Yes, there's no 16-bit support in Irfanview, but I left this issue aside meanwhile.
3. Wherever I do have to choose and apply color correction, I want this to be as fast and convenient as possible. “Fast” in my case applies mostly to the “attended” time – the time I myself have to spend near computer to make the corrections. “Convenient” in my case means minimal number of movements and all of them ergonomics-friendly, and not waiting near computer for longer-than-instant operations. And, again, in my case, “ergonomics-friendly” means NO MOUSE unless absolutely needed. **The key to the solution was the fact that I always use a pretty limited set of color corrections. So I decided to group**

images by the kind of corrections they require, then apply all the chosen corrections in one batch. Note, that in order to decide on a correction, I look at JPEG ("preview"), while the correction should be applied during the conversion of the raw image.

4. How to "group images by the kind of corrections they require" in a convenient and efficient way. The solution is to distribute ("sort") the previews between directories, each directory linked to a conversion with certain kind of color correction. These directories are called *sorting directories* or *correction directories* further.
5. TODO?

Requirements.

Computer system.

LazyConv in its current form could be run only on MS-Windows – based systems.

Tested on the following three configurations:

1. (Desktop) Pentium 4 2400GHz, 768Mb RAM, WinXP Professional
2. (Laptop) Centrino 1.76GHz, 512Mb RAM, WinXP Home
3. (Desktop) Core 2 Duo 2.4GHz, 2Gb RAM, Windows Vista Home Premium

Disk space required for the full installation of LazyConv is ~200 Mb plus Irfanview, whatever the latter takes.

Disk space required for the usage of LazyConv depends on the numbers and size of the raw files being processed. I must admit, disk space is heavily used by LazyConv, so plan to keep some ~10Gb available if you constantly use LazyConv for large bunches of images.

User knowledge.

The use of LazyConv **does not require**:

- knowledge of TCL or any other programming language

The use of LazyConv **does require**:

- basic familiarity with command-line computer interface
- user-level knowledge of Windows file system – what is it file, what is it directory, what is it path, what is it filename extension, basic file-manipulation commands (like dir, copy, md, rename, erase), etc.
- understanding of photo file formats – TIFF, JPEG
- understanding of basic color-correction methods: curves, gamma, contrast, etc.

Simple session.

This must-read chapter illustrates typical image-processing session; reading it gives an ultimate feel of LazyConv. It has most of what you should know in order to use LazyConv,

and most of this information isn't duplicated in this guide.

It's recommended to read this section twice:

> First time “off-line” - without trying to run LazyConv in parallel; just concentrate on the presented sequence of commands and their outcomes; this chapter requires no prerequisites to understand the idea of LazyConv.
>
> Second time, after you read “Installation of LazyConv” and followed its instructions, use this section as hands-on tutorial; copy-paste the commands and they will work.

LazyConv is a command-line based tool. While this could look unfamiliar for most photo enthusiasts, years of development of the computer industry showed that command line is the most efficient way of working with computer. And for processing hundreds of images nothing is as important.

The commands shown below are executed from within TCL shell (I'm using TkCon for the purpose).

Of course, common file-manipulation commands (copy, move/rename, make-directory) could be run by other means available in Windows. When processing my own photos, I'm using Irfanview for viewing and sorting images, and it's configured in LazyConv for these purposes. See “Using Irfanview for image viewing and sorting“ for useful tips.

Note file-path convention being used: “/” (slash) delimits nested directories' names – this conforms standards of TCL language and UNIX operating system. And thus it works this way in LazyConv, even when it runs on Windows where “\” (backslash) serves as a directory delimiter.

Commands that belong to LazyConv itself are shown in **bold** font.

So, enter a TCL shell of your choice, create a new directory and go into it. Let's assume this directory is C:/ANY/LazyConv/Doc/Demo/.

Note that full paths are used throughout the session – relative path approach won't work.

Create a directory for input (raw) files and bring them there:

```
Demo> file mkdir C:/ANY/LazyConv/Doc/Demo/INPUTS

Demo> file copy C:/ANY/LazyConv/tcl/work/Run/Raw/MINOLTA/PICT1080.MRW
   C:/ANY/LazyConv/tcl/work/Run/Raw/MINOLTA/PICT1889.MRW
   C:/ANY/LazyConv/tcl/work/Run/Raw/MINOLTA/PICT1998.MRW
   C:/ANY/LazyConv/tcl/work/Run/Raw/MINOLTA/PICT2011.MRW
   C:/ANY/LazyConv/Doc/Demo/INPUTS/
```

Load LazyConv code into TCL interpreter; the example below configures LazyConv with Irfanview as raw converter:

```
Demo> source C:/ANY/LazyConv/tcl/Work/setup_rconv.tcl

.............

-I- Convert command definitions taken from
   'C:/ANY/LazyConv/TCL/Work/tool_wrap_main/wrap_rconv_iview.tcl' .
```

-I- Viewer command definitions taken from
'C:/ANY/LazyConv/TCL/Work/tool_wrap_main/wrap_view_iview.tcl' .

.....

Demo>

Build directory structure required for one session. The following switches should be given to "**run_setup_work_area**" command:

- "-inp_dirs_root" tells where the inputs are
- "-work_dirs_root" tells what directory to use for output
- "-origin" tells the kind of input files in this session (like Minolta-raw, TIFF, JPEG, Nikon-raw, etc.)

Demo> **run_setup_work_area -origin MINOLTA -inp_dirs_root *C:/ANY/LazyConv/Doc/Demo/INPUTS/* -work_dirs_root *C:/ANY/LazyConv/Doc/Demo/***

.....

-I- Input config printed into 'C:/ANY/LazyConv/Doc/Demo/inp_config.tcl'

-I- Output config printed into 'C:/ANY/LazyConv/Doc/Demo/out_config.tcl'

-I- Created destination directory 'C:/ANY/LazyConv/Doc/Demo/Out'.

.....

Demo>

Instruct the converter to use directory *C:/ANY/LazyConv/Doc/Demo/* as a working area; it should already contain files inp_config.tcl and out_config.tcl that specify required inputs' and outputs' definitions. For example, input files will be looked for under *C:/ANY/LazyConv/Doc/Demo/INPUTS/* . Note that you may come back to work with the same directory multiple times – after shutting down the computer or just exiting the TCL shell; each time you should issue the "set_work_area" command like below.

Demo> **set_work_area *C:/ANY/LazyConv/Doc/Demo/***

-I- Configuring inputs from 'C:/ANY/LazyConv/Doc/Demo//inp_config.tcl'

-I- Configuring outputs from 'C:/ANY/LazyConv/Doc/Demo//out_config.tcl'

.....

Demo>

Let's take a look at the images in our working area before the conversion actually starts. One should expect four .mrw (Minolta raw) files under INPUTS/ directory, and no more images.

Demo> dir *C:/ANY/LazyConv/Doc/Demo*

C:/ANY/LazyConv/Doc/Demo:

INPUTS Out inp_config.tcl out_config.tcl

Demo> dir *C:/ANY/LazyConv/Doc/Demo/INPUTS/*

INPUTS/:

```
        PICT1080.MRW  PICT1889.MRW  PICT1998.MRW  PICT2011.MRW
Demo> dir out/
        out/:
Demo>
```

Make initial *preview*s of all the raw-s – with default color settings. These *preview*s are JPEG images with reasonable file-size that enable the user to asses how the ultimate photos will look like. The previews are used to:

- decide whether to keep particular image; just delete the preview if you don't need this image
- guess what (predefined) color correction it is worth to apply to the image; move or copy the preview into an image-sorting directory linked to this correction; further you will be able to compare the original preview with the one that undergone that correction

Notice the messages being printed while the command executes. Since this is the first conversion session being run in this work area, it creates image-sorting directories as a by-product.

"**run_preview_all**" is a long-running command; console screen could stay frozen until it's finished.

```
Demo> run_preview_all
-I- Configuring inputs from 'C:/ANY/LazyConv/Doc/Demo//inp_config.tcl'
-I- Configuring outputs from 'C:/ANY/LazyConv/Doc/Demo//out_config.tcl'
-I- Re-mapped dir '.'          to command 'CONVERT_ROTATE_MRW_DEFAULT'
-I- Defined 18 mapped conversion commands
-I- Created correction/conversion input directory 'C:/ANY/LazyConv/Doc/Demo/Out/DoContrPlus1'.
-I- Created correction/conversion input directory 'C:/ANY/LazyConv/Doc/Demo/Out/Manual'.
.....
-I- Created correction/conversion input directory 'C:/ANY/LazyConv/Doc/Demo/Out/KeepAsIs'.
-I- Running in PREVIEW MODE - all input images will be converted by
   'CONVERT_ROTATE_MRW_DEFAULT' command.
-I- -------- Session started at 14/08/06-14:41:38 -----------
-I- CONVERT_ROTATE_MRW_DEFAULT          C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1080.MRW
            -> C:/ANY/LazyConv/Doc/Demo/Out/PICT1080_ivrdef.jpg
-I- CONVERT_ROTATE_MRW_DEFAULT          C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1889.MRW
            -> C:/ANY/LazyConv/Doc/Demo/Out/PICT1889_ivrdef.jpg
-I- CONVERT_ROTATE_MRW_DEFAULT          C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1998.MRW
            -> C:/ANY/LazyConv/Doc/Demo/Out/PICT1998_ivrdef.jpg
-I- CONVERT_ROTATE_MRW_DEFAULT          C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT2011.MRW
            -> C:/ANY/LazyConv/Doc/Demo/Out/PICT2011_ivrdef.jpg
-I- ==== Done. Processed 4 files. Skipped 0 already existing files. ====
```

```
Demo>
```

Now observe the previews and image-sorting directories being created; the latter are shown with ***bold italic*** font below:

```
Demo> dir Out/
Out/:
```

DoBrightP01	***DoBrightP1***	***DoBrightP2***	***DoContrMinus1***
DoContrMinus2	***DoContrPlus1***	***DoContrPlus1Dark1***	***DoContrPlus2***
DoContrPlus2Dark1	***DoDark***	***DoDark1ContrPlus1***	***DoDark1ContrPlus2***
DoDefault	***DoDesatBright***	***DoDesatBrightP1***	***KeepAsIs***
Manual	PICT1080_ivrdef.jpg	PICT1889_ivrdef.jpg	PICT1998_ivrdef.jpg

```
PICT2011_ivrdef.jpg
```

As expected, we got four preview images for four raw files. Name of a preview is assembled out of name-base (“PICTnnnn”, taken from its raw file), a suffix telling what conversion command was used to create this preview (“_ivrdef” for default command) and standard extension for JPEG format (“.jpg”).

The previews are placed in directory Out/, which is the *default output directory* for the LazyConv session.

The list of *image-sorting directories* depends on user-controllable LazyConv configuration, so you may obtain a different one. The name of a directory should reflect the function of its command; in my case “DoDark” means make the image darker, and so on. The *image-sorting directories* reside under the *default output directory*.

You may learn about conversion commands being configured by issuing **run_describe_all_commands**, **run_describe_mapped_commands** and “**run_describe_command** <command_name>” at the LazyConv prompt (the output of **run_describe_all_commands** is too long, so I don't show it all):

```
Demo> run_describe_all_commands
-I- Command CONVERT_MRW_CONTRM1:        no directory      - performs raw conversion with a little
    less contrasty output
-I- Command CONVERT_MRW_DARKER:         no directory      - performs raw conversion with darker
    output
-I- Command CONVERT_ROTATE_MRW_CONTRM1:    directory - DoContrMinus1/       - performs raw
    conversion with a little less contrasty output and rotation to match the preview image
.....
-I- Command CONVERT_ROTATE_MRW_DEFAULT:    directory : "DoDefault/" : "./"      - performs raw
    conversion with default colors and rotation to match the preview image
-I- Command CONVERT_ROTATE_MRW_DESAT_BRIGHT:      directory - DoDesatBright/       -
    performs raw conversion while trying to preserve (desaturate) highlights; rotates output to match the
    preview image
-I- Command CONVERT_ROTATE_MRW_DARKER:      directory - DoDark/       - performs raw
    conversion with darker output and rotation to match the preview image
```

```
Demo>
```

“**run_describe_all_commands**” prints short description for each conversion command configured in your LazyConv installation. It includes nickname you gave to this command (like CONVERT_ROTATE_MRW_DEFAULT), directory linked to it, and the functional description.

You may notice that some conversion commands don't have an associated directory; these commands do exist in the system, but won't be accessible.

Pay attention that CONVERT_ROTATE_MRW_DEFAULT has two directories linked to it, the default output directory (.) and another explicit directory where you should put an image's preview to cause creating preview with default colors for this image. This is useful if you mistakenly deleted the default preview.

Use “**run_describe_mapped_commands**” to see only commands that do have associated sorting directory.

```
Demo> run_describe_command CONVERT_ROTATE_MRW_DARKER
-I- Command CONVERT_ROTATE_MRW_DARKER:      directory - DoDark/      - performs raw
    conversion with darker output and rotation to match the preview image
Demo>
```

Now let's perform a kind of image sorting:

- image PICT2011 appeared non-interesting, delete its preview
- image PICT1080 has blown highlights, pass it through a command that tries to fix them
- image PICT1998 is too low on contrast; pass it through a contrast-enhancement command
- image PICT1889 requires non-trivial treatment; “schedule” it for manual conversion

Note that some images are moved into sorting directories, (*file rename*) while others are copied/duplicated (*file copy*); usually it's better to move images, unless you want to try several different corrections.

```
Demo> file delete Out/PICT2011_ivrdef.jpg
Demo> file copy Out/PICT1080_ivrdef.jpg out/DoDesatBright/
Demo> file copy Out/PICT1998_ivrdef.jpg out/DoContrPlus1/
Demo> file rename Out/PICT1889_ivrdef.jpg out/Manual/
Demo>
```

So, here is the arrangement of previews to trigger the corrections we decided upon:

```
Demo> foreach f [lsort [glob {out/*.jpg}]] {puts $f}
        Out/PICT1080_ivrdef.jpg
        Out/PICT1998_ivrdef.jpg
Demo> foreach f [lsort [glob {Out/*/*.jpg}]] {puts $f}
```

```
        Out/DoContrPlus1/PICT1998_ivrdef.jpg
        Out/DoDesatBright/PICT1080_ivrdef.jpg
        Out/Manual/PICT1889_ivrdef.jpg
    Demo>
```

The “**run_correction**” command below will pass over all previews in image-sorting directories and apply to each preview a command linked to its directory.

“**run_correction**” is a long-running command; console screen could stay frozen until it's finished.

```
    Demo> run_correction
    -I- Configuring inputs from 'C:/ANY/LazyConv/Doc/Demo//inp_config.tcl'
    -I- Configuring outputs from 'C:/ANY/LazyConv/Doc/Demo//out_config.tcl'
    ..........
    -I- Total 3 file[s] in 3 directories
    -I- -------- Session started at 14/08/06-15:02:59 -----------
    -I- Input='PICT1080.MRW'            Previews(1)
       ='C:/ANY/LazyConv/Doc/Demo/Out/DoDesatBright/PICT1080_ivrdef.jpg'
    -I- CONVERT_ROTATE_MRW_DESAT_BRIGHT C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1080.MRW
            -> C:/ANY/LazyConv/Doc/Demo/Out/PICT1080_ivrdebright.jpg
    -I- Input='PICT1889.MRW'            Previews(1)
       ='C:/ANY/LazyConv/Doc/Demo/Out/Manual/PICT1889_ivrdef.jpg'
    -I- COPY_INPUT_FOR_MANUAL_PROC  C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1889.MRW ->
       <Unknown-OutPath>
    -I- Input='PICT1998.MRW'            Previews(1)
       ='C:/ANY/LazyConv/Doc/Demo/Out/DoContrPlus1/PICT1998_ivrdef.jpg'
    -I- CONVERT_ROTATE_MRW_CONTRP1          C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1998.MRW
            -> C:/ANY/LazyConv/Doc/Demo/Out/PICT1998_ivrcnp1.jpg
    -I- ==== Done. Processed 3 files. Skipped 0 already existing files. ====
    Demo>
```

And here is the outcome; results of color correction should be found in “Out/” directory; the previews that triggered color corrections are moved to “out/” directory as well (if they were left in image-sorting directories, another round of identical color corrections would be attempted).

```
    Demo> foreach f [lsort [glob {Out/*.jpg}]] {puts $f}
        Out/PICT1080_ivrdebright.jpg
        Out/PICT1080_ivrdef.jpg
        Out/PICT1998_ivrcnp1.jpg
        Out/PICT1998_ivrdef.jpg
    Demo> foreach f [lsort [glob {Out/*/*.*}]] {puts $f}
        Out/Manual/PICT1889.MRW
```

```
        Out/Manual/PICT1889_ivrdef.jpg
    Demo>
```

As you see, three images have undergone a processing:

- PICT1080 and PICT1998 passed through color-correction commands – now you have two different previews of each, their name-suffixes reflect the commands being applied
- the input (raw) file for PICT1889 was copied into “Manual/” sorting directory (where its preview stays); this is to conveniently collect together all images that require manual processing

The next step is to compare different previews of the same image and choose the best one. (Of course one may decide to try other corrections before taking the final choice.) Look at the previews and delete unneeded ones. (Though nothing prevents you from keeping different versions altogether.)

```
Demo> file delete Out/PICT1080_ivrdef.jpg
Demo> file delete Out/PICT1998_ivrcnp1.jpg
Demo>
```

Now, the “**run_conversion**” command will create ultimate TIFF images that match previews you chose to keep. It will base its choices of color corrections on the name-suffixes of these previews.

“**run_conversion**” is a long-running command; console screen could stay frozen until it's finished.

```
Demo> run_conversion
-I- Configuring inputs from 'C:/ANY/LazyConv/Doc/Demo//inp_config.tcl'
-I- Configuring outputs from 'C:/ANY/LazyConv/Doc/Demo//out_config.tcl'
.....
-I- Created final output directory C:/ANY/LazyConv/Doc/Demo/Out/Final
.....
-I- -------- Session started at 14/08/06-17:26:03 -----------
-I- Input='PICT1080.MRW'          Previews(1)
   ='C:/ANY/LazyConv/Doc/Demo/Out/PICT1080_ivrdebright.jpg'
-I- CONVERT_ROTATE_MRW_DESAT_BRIGHT C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1080.MRW
          -> C:/ANY/LazyConv/Doc/Demo/Out/Final/PICT1080_ivrdebright.tif
-I- Input='PICT1998.MRW'          Previews(1)='C:/ANY/LazyConv/Doc/Demo/Out/PICT1998_ivrdef.jpg'
-I- CONVERT_ROTATE_MRW_DEFAULT          C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT1998.MRW
          -> C:/ANY/LazyConv/Doc/Demo/Out/Final/PICT1998_ivrdef.tif
-I- ==== Done. Processed 2 files. Skipped 0 already existing files. ====
Demo>
```

So, we expect two final TIFF files to be created. They are placed in “Out/Final/” directory created by this (first) invocation of “**run_conversion**” command.

```
Demo> foreach f [lsort [glob {Out/Final/*.tif}]] {puts $f}
        Out/Final/PICT1080_ivrdebright.tif
        Out/Final/PICT1998_ivrdef.tif
Demo>
```

These TIFFs were “dictated” by the following previews found by “**run_conversion**” in the “out/” directory:

```
Demo> foreach f [lsort [glob {Out/*.jpg}]] {puts $f}
        Out/PICT1080_ivrdebright.jpg
        Out/PICT1998_ivrdef.jpg
```

The preview in “Out/Manual/” directory was ignored – we expect that this image will be manually converted outside of LazyConv.

```
Demo> foreach f [lsort [glob {Out/*/*.jpg}]] {puts $f}
        Out/Manual/PICT1889_ivrdef.jpg
```

And the last stage is to clean-up unneeded input (raw) files. You know which previews are missing (since you chose to delete them), now the “**run_process_unlisted**” command will rename input (raw) files for which no previews exist anywhere **under** our “out/” directory.

```
Demo> foreach f [lsort [glob {INPUTS/*.mrw}]] {puts $f}
        INPUTS/PICT1080.MRW
        INPUTS/PICT1889.MRW
        INPUTS/PICT1998.MRW
        INPUTS/PICT2011.MRW
Demo>
```

You know, there's no preview for PICT2011.

```
Demo> run_process_unlisted
-I- Configuring inputs from 'C:/ANY/LazyConv/Doc/Demo//inp_config.tcl'
-I- Configuring outputs from 'C:/ANY/LazyConv/Doc/Demo//out_config.tcl'
.....
-I- Running in PROCESS-UNLISTED-FILES MODE - a 'RENAME_UNLISTED_FILE' command will be
    applied to input files for which no preview images exist.
.....
-I- -------- Session started at 14/08/06-17:28:47 -----------
-I- RENAME_UNLISTED_FILE      C:/ANY/LazyConv/Doc/Demo/INPUTS/PICT2011.MRW -> <Unknown-
    OutPath>
-I- ==== Done. Processed 1 files. Skipped 0 already existing files. ====
```

```
Demo>
```

Notice PICT2011 received "_ToDelete" suffix in its name. Now you can delete all such files at once by "erase *ToDelete" MSDOS command. Or you may choose to move them away into some other directory by means of the same wildcard.

```
Demo> foreach f [lsort [glob {INPUTS/*.mrw}]] {puts $f}
    INPUTS/PICT1080.MRW
    INPUTS/PICT1889.MRW
    INPUTS/PICT1998.MRW
    INPUTS/PICT2011_ToDelete.MRW
Demo>
```

Here is how you can delete the unnecessary inputs from inside LazyConv (TCL shell):

```
Demo> foreach f [glob {INPUTS/*ToDelete*.mrw}] {file delete $f}
```

And this is the end of the sample LazyConv session.

Installation of LazyConv.

Distribution contents.

LazyConv comes in an archive file containing the following directories:

- Doc - various documentation including the User Guide
- freewrap62 - standalone TCL interpreter "freewrap" renamed into "TclInterp.exe"
- IM_EXE - ImageMagick tools (standalone executables): convert, identify
- TCL - TCL source code, mostly that of LazyConv

In case your computer already has some of the tools installed, you may use them instead of those you get in LazyConv distribution.

As dcraw engine is optional, it is not included into the LazyConv distribution; you my download it separately either from LazyConv web site, or from any other appropriate source.

Installation steps.

(Absolute paths below should be treated as examples. Relative paths are assumed to start from the root directory of LazyConv.zip archive (LazyConv/).

1. Install Irfanview if you don't already have it. This is the only mandatory component of LazyConv that is not included in the distribution archive. Separate installation of the rest of the tools is optional and not recommended for anybody but advanced user or computer professional.

   You can download it for free from www.irfanview.com .

2. Unpack the LazyConv.zip archive in the directory of your choice.

In the examples below directory C:\ANY\ is used, so that the resulting LazyConv directory is C:\ANY\LazyConv\

The commands in these examples assumed to be typed in one MS-DOS command window (Start -> Programs ->Accessories -> Command Prompt).

```
C:\> set ICDir=C:\ANY\LazyConv
C:\> cd /D %ICDir%
C:\ANY\LazyConv> dir
      .....
       Directory of C:\ANY\LazyConv
      .....
23/08/2006  11:09 PM    <DIR>          Doc
14/04/2007  11:09 PM    <DIR>          EXIV2
23/08/2006  11:09 PM    <DIR>          freewrap62
23/08/2006  11:09 PM    <DIR>          IM_EXE
23/08/2006  11:09 PM    <DIR>          TCL
```

3. You received ImageMagick executables as a part of LazyConv archive – under IM_EXE\ . Though you may choose to install ImageMagick separately.

   ImageMagick is available at www.imagemagick.org

4. You received standalone TCL interpreter as a part of LazyConv archive. The file is freewrap62\TclInterp.exe, and it's the same as freewrap62\freewrap.exe but renamed. You may prefer to download and install TCL interpreter of your choice instead.

   Freewrap itself could be downloaded from freewrap.sourceforge.net

   A full TCL installation is available at www.tcl.tk; get it if you plan to use TCL for something beyond running LazyConv.

5. You received EXIV2 standalone EXIF reader as a part of LazyConv distribution – under EXIV2 directory. This tool is not mandatory for the current LazyConv version.

   EXIV2 could be downloaded from www.exiv2.org

6. TCL directory contains both LazyConv own source-code area (TCL\Work\) and the following tools that you may want to download separately:

   - TCL\Console\tkcon.tcl is a convenient TCL console; I'd recommend using it instead of the native one; see invocation example in “Starting TCL interpreter”

     the source is tkcon.sourceforge.net

   - Directory TCL\Libs\ contains standard TCL extensions that are crucial for LazyConv – Tclx and TclLib

     You obtain them automatically if you take a full TCL installation at www.tcl.tk .

     They don't come together with FreeWrap interpreter.

The path to these libraries is explicitly configured in LazyConv (see “Update TCL source-code search paths” below), so it's not a problem to have even multiple versions of them on the computer

7. If you want to use dcraw as the raw conversion engine, download it separately:
   - either from www.lazyconv.com (private version with channel multipliers, see DCRAW based “snapshot” raw conversion commands)
   - or from any official source, see http://cybercom.net/~dcoffin/dcraw/ for the links

     you will have to modify dcraw wrapper a bit – choose a version of command-line flags without channel multipliers, see DCRAW based “snapshot” raw conversion commands)
8. Update external tools' paths in TCL\Work\cnv_config\ext_tools.tcl to match your system (note “/” being used as delimiter since it's TCL). If you use the tools from LazyConv archive, only the **bold** parts of the paths require changes:

   C:\ANY\LazyConv> write TCL\Work\cnv_config\ext_tools.tcl

   (tip: if you copy the lines from this document, use “Paste Special”->”Unformatted Text” to insert them into the settings-files you open in WordPad)
   - Irfanview executable path

     variable IRFANVIEW "**C:/Program Files/IrfanView/i_view32.exe**"
   - ImageMagick executable paths

     variable IMCONVERT "**C:/ANY**/LazyConv/IM_EXE/convert.exe"

     variable IMIDENTIFY "**C:/ANY**/LazyConv/IM_EXE/identify.exe"
   - dcraw executable path (optional)

     variable DCRAW "**C:/ANY/CWork/DCRAW**/dcraw.exe"
   - Exiv2 executable path (optional)

     variable EXIV2 "**C:/ANY**/LazyConv/EXIV2/exiv2.exe"
   - path of directory for temporary files

     variable TMPDIR "**C:/ANY/LazyConv**"

     if you have a RAM disk, use it for the temporary files
9. Update TCL source-code search paths in TCL/Work/setup.tcl to match your LazyConv installation; usually only the **bold** parts of the paths require changes here

   C:\ANY\LazyConv> write TCL\Work\setup.tcl
   - root directory for TCL source code of LazyConv

     set OK_TCLSRC_ROOT "**C:/ANY/**LazyConv**/**TCL/Work"
   - root directory for TCL standard extension libraries

set OK_TCLSTDEXT_ROOT "**C:/ANY/**LazyConv**/**TCL/Libs"

- root directory for converter internal use

  set OK_CONV_WORK_DIR "**C:/ANY/**LazyConv**/**TCL/Work"

10. Update test-related paths in TCL/Work/cnv_test/config_test.tcl to match your LazyConv installation; usually only the **bold** parts of the paths require changes here

    C:\ANY\LazyConv> write TCL\Work\cnv_test\config_test.tcl

    - directory under which sample input files are stored

      variable TEST_INPUT_IMG_ROOT "**C:/ANY/LazyConv/TCL/Work/Run/Raw**"

    - directory under which checksums for sample output files are stored

      variable DB_OUTDIR "**C:/ANY**/LazyConv/TCL/Work/Run/TestOut"

11. Update Irfanview settings (directory for each preset) - delete old directories to enforce automatic creation of the new ones

    - Irfanview settings' directories are located under TCL/Work/IView_Settings

      either delete all directories there, or move them to anywhere; next time LazyConv runs, it will create new Irfanview settings' directories that match your system configuration

      C:\ANY\LazyConv> rmdir /S/Q TCL\Work\IView_Settings

Now LazyConv should be ready for use.

Using LazyConv.

Most of this material is already introduced in “Simple session”; but here more details are provided.

Starting TCL interpreter.

Since LazyConv is TCL-based tool, you need to obtain TCL interpreter command prompt, and then run LazyConv from it.

If you installed TCL interpreter yourself, you can probably skip this section.

To start the native console of freewrap (supplied with LazyConv):

Run TclInterp.exe from MS-DOS prompt or "Run" menu, or shortcut:

C:\ANY\LazyConv\freewrap62\TclInterp.exe

two windows appear; the one that says “Console” in its title bar is your TCL interpreter

To start the more advanced and convenient “tkcon” console (supplied with LazyConv):

Run TclInterp.exe from MS-DOS prompt or "Run" menu, or shortcut while providing it with the path to the console TCL code (tkcon.tcl):

C:\ANY\LazyConv\freewrap62\TclInterp.exe
C:\ANY\LazyConv\TCL\Console\tkcon.tcl

the new TCL-interpreter window should appear.

I'd recommend creating a shortcut for the interpreter invocation; you'll have to run it again for each conversion session.

Choosing the right conversion scheme.

The conversion scheme here means conversion engine (Irfanview, dcraw, etc.) together with LazyConv wrapper. This choice affects what set of conversion/correction commands will be available in your LazyConv session.

Common for all engine+wrapper pairs is the following convention:

- there always is a default conversion command and it is applied by run_preview_all
- effects of other conversion/correction commands are relative to the default command; for example “dark” means the output will be darker than that of the default command

The difference is best explained as lying in the definition of the default command. Currently (as for version 1.2) two approaches are supported:

1. “Snapshot” type - the default command tries to render any image as “correctly” exposed.
    - most suitable for snapshots, vacation photos, event journalism
    - only extremely under/overexposed images turn out dark/bright
    - implemented by the following LazyConv wrappers:
        - setup_rconv.tcl - see “Irfanview based raw conversion commands”
        - setup_rconv_dcraw.tcl - see “DCRAW based “snapshot” raw conversion commands”
2. “Real” type - the default command strictly follows the shooting exposure.
    - most suitable for art work
    - provides closest match to in-camera JPEG and image preview
    - implemented by the following LazyConv wrappers:
        - setup_rconv_cam_dcraw.tcl - see “DCRAW based “real” raw conversion commands”

Setting-up LazyConv for a session.

From TCL prompt run converter setup - “source” one of the following setup files:

- when processing camera-raw files (like .mrw), “source” TCL/Work/**setup_rconv.tcl** to configure Irfanview as raw converter, or either TCL/Work/**setup_rconv_dcraw.tcl** or TCL/Work/**setup_rconv_cam_dcraw.tcl** to configure dcraw as raw converter
- file TCL/Work/**setup_iconv.tcl** when processing standard images (like .tif):

Use the info provided in “Choosing the right conversion scheme” to make the above choice.

Example (raw images, Irfanview as engine):

source C:/ANY/LazyConv/TCL/Work/setup_rconv.tcl

Now your TCL interpreter will recognize LazyConv commands.

You should create a *work-area* for each set of photos to be processed altogether. Work-area consists of a group of directories expected by LazyConv, as well as input- and output configuration files fitted for this specific session (inp_config.tcl and out_config.tcl).

First, create the work-area root directory:

file mkdir C:/ANY/LazyConv/MyRun

Normally, you should then create a directory for input files, but in the example here we'll use the existing one.

Creation of a work-area in existing directory is performed by the following command:

run_setup_work_area \
-origin <MINOLTA|TIFF|JPEG|...> \
-inp_dirs_root <root path of input images' directories> \
-work_dirs_root <root path of output/sort directories> \
[-sort_dir_names <list of sort directories' names>] \
[-clean <yes|no>]

- -inp_dirs_root tells the path to directory under which the input files reside, either right in this directory, or in its subdirectories
- -work_dirs_root tells what is the root directory for output
  - without the “-sort_dir_names” switch, there will be a single group of color-correction directories under <work_dirs_root>/Out/
  - with the “-sort_dir_names” switch, you specify several such groups
- -sort_dir_names (optional) tells names (not paths) for root directories of sorting groups

  if you want to pre-sort images for some reason (by shooting location, date, subjects, whatever), this switch forces creation of several groups of output directories; see “Using two groups of image-sorting directories”
- -origin tells the kind of input files in this session (see “Supported raw-file origins”)

  example: BMP, CANON_CR2, JPEG, MINOLTA, NIKON, TIFF
- -clean <yes|no> (optional, default is “no”) tells whether to clean output directory and input/output configuration files

From TCL prompt run work-area creation:

run_setup_work_area -origin MINOLTA -inp_dirs_root C:/ANY/LazyConv/tcl/Work/Run/Raw/ -work_dirs_root C:/ANY/LazyConv/MyRun

If you attempt to run the above command twice, second time without “-clean yes”, existing configuration files will not be overridden. So if you don't delete anything from the work area, second invocation of **run_setup_work_area** will have no effect.

From TCL prompt set work-area directory:

set_work_area C:/ANY/LazyConv/MyRun

- You should issue “set_work_area” command each time you come-back to work with this directory (like when you open a new TCL shell after switching on the computer).

Now you can start processing your images in this area.

Supported raw-file origins.

LazyConv “inherits” the list of supported camera-raw formats from the raw-conversion engine it uses. The current version of LazyConv works with Irfanview (tested with versions 3.98 to 4.10) and dcraw; you may find the details on supported cameras on www.irfanview.com and www.cybercom.net/~dcoffin/dcraw/ accordingly.

But there's a little complication – LazyConv needs to know name patterns for the raw files, otherwise **run_setup_work_area** cannot build the correct session configuration.

Here is the table telling what origins are already included, and which of them I actually tested.

Origin name	Cameras	Tested?
CANON_CR2	EOS350D, EOS300D, etc?	yes
CANON_CRW	?	no
KODAK	DSLR/c, DSLR/n, ?	no
MINOLTA	A2, 5D, 7D, ?	yes
NIKON	D100, D50, D70, ?	yes
OLYMPUS	E1, E500, E300, ?	no
PENTAX	*istD, *istDS, ?	no
SIGMA	SD9, SD10	no
SONY_SR2	DSC-R1	no
Sony_ARW	Alpha 100	yes

Again, in most cases the camera is supported, what could be missing or incorrect, is the name-pattern definition. See “Adding or fixing raw-file origins” on how to fix this.

Using the latest official release of dcraw as the raw conversion engine guarantees maximal number of cameras being supported.

Irfanview, while easier to install, isn't updated as frequently, so if you have a relatively new camera, you should go for dcraw.

Anyway it's a good idea first to try opening a representative RAW with Irfanview (double-click on the RAW image); if it says format is unrecognized, configure LazyConv to work with dcraw.

Troubleshooting.

If you get error message like below, check your TCL-related path settings in TCL/Work/setup.tcl

.....

---- Sourcing C:/ANY/LazyConv/TCL/Work/ok_utils/common.tcl ----

can't find package cksum

- In this case, the " OK_TCLSTDEXT_ROOT" variable wasn't set properly (see "Installation steps"). If fixing this doesn't help, download and install tcllib – go to www.tcl.tk and follow their instructions.

If you try running Irfanview RAW conversion, and dialog boxes pop-up saying RAW images are unreadable, your version of Irfanview doesn't support your camera.

- Download, install and configure a compatible version of dcraw as the RAW conversion engine.

Main commands reference.

After you successfully followed the setup procedure in "Setting-up LazyConv for a session", you may run the following LazyConv commands from TCL prompt:

run_preview_all - make default-color previews for all available input images

- name of a preview for input (raw) file PICTxxxx.<EXT> will be PICTxxx_<default_suffix>.JPG

  example: PICT1080.MRW -> PICT1080_ivrdef.JPG corresponds to the default settings in LazyConv (as it ships)

- preview images are placed in the default output directory – the one specified through the "-inp_dirs_root" switch to "**run_setup_work_area**"
- if a file with the name equal to a specific preview image already exists, it will not be overwritten

run_browse_previews_in_dir <directory_path> - a convenient way to run image viewer configured in LazyConv; after obtaining the previews you should browse them and decide what to do with each; see "Sorting images for color corrections and more"

- this version of LazyConv comes with Irfanview as a single supported viewer
- see "Using Irfanview for image viewing and sorting" for relevant Irfanview usage tips

- only images in the specified directory will be shown; not in the subdirectories
- the console becomes non-responsive until the viewer is exited

run_correction - for each preview image found in a correction directory, run a command associated with this directory

- nothing will happen for previews located in the default output directory
- LazyConv supports two special pseudo correction directories:
  - Manual/ input files for previews found there will be copied into this directory too
  - KeepAsIs/ previews found there are ignored by **run_correction**, but encountered by **run_conversion**
- if a file with the name equal to a specific corrected preview image already exists, it will not be overwritten

run_conversion - for each preview found either in default output directory, or in correction directory (except Manual/), create the final image with color settings defined by the preview's suffix

- the final images are stored in subdirectory Final/ of the default output directory
- final image differs from the corresponding preview by the output format – usually the preview is JPEG, while final image is TIFF; both are configurable
- if there are several previews for one input, there will be several final images
- if a file with the name equal to a specific final image already exists, it will not be overwritten

run_process_unlisted - all input (raw) images for which no previews exist will be renamed – this is to facilitate efficient cleanup

- the previews for this purpose are looked for both in default output directory, and in all the correction/sorting directories (including Manual/)
- final images are not considered by **run_process_unlisted**; do not run it if you deleted the previews!

run_remake_wb_ovrd_previews - recreates previews for which white-balance overrides are requested; see “Overriding white balance”.

Sorting images for color corrections and more.

This is nearly the only thing the user does manually in LazyConv. Sometimes several sort-images + **run_correction** iterations are required until the user is satisfied.

To obtain the sorting/correction directories configured on your site together with associated commands, use LazyConv's command **run_describe_mapped_commands**:

MyRun> **run_describe_mapped_commands**

-I- Command CONVERT_ROTATE_MRW_CONTRP1: directory : "**DoContrPlus1**/" - performs raw conversion with a little more contrasty output and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_CONTRP2: directory : "**DoContrPlus2**/" - performs raw conversion with much more contrasty output and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_BRIGHTP1: directory : "**DoBrightP1**/" - performs raw conversion with a little brighter output and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_BRIGHTP2: directory : "**DoBrightP2**/" - performs raw conversion with significantly brighter output; rotates output to match the preview image

-I- Command CONVERT_ROTATE_MRW_DARK1CONTRP1: directory : "**DoDark1ContrPlus1**/" - performs raw conversion with a little more darker and contrasty output, and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_CONTRP1DARK1: directory : "**DoContrPlus1Dark1**/" - performs raw conversion with a little more contrasty and darker output, and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_DARK1CONTRP2: directory : "**DoDark1ContrPlus2**/" - performs raw conversion with darker and much more contrasty output, and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_CONTRM1: directory : "**DoContrMinus1**/" - performs raw conversion with a little less contrasty output and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_CONTRM2: directory : "**DoContrMinus2**/" - performs raw conversion with much less contrasty output and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_DEFAULT: directory : "**DoDefault**/" : "**./**" - performs raw conversion with default colors and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_DESAT_BRIGHT: directory : "**DoDesatBright**/" - performs raw conversion while trying to preserve (desaturate) highlights; rotates output to match the preview image

-I- Command CONVERT_ROTATE_MRW_BRIGHTP01: directory : "**DoBrightP01**/" - performs raw conversion with brighter low- and midtones; rotates output to match the preview image

-I- Command CONVERT_ROTATE_MRW_DESAT_BRIGHTP1: directory : "**DoDesatBrightP1**/" - performs raw conversion while trying to preserve (desaturate) highlights, but the output is slightly brighter; rotates output to match the preview image

-I- Command CONVERT_ROTATE_MRW_DARKER: directory : "**DoDark**/" - performs raw conversion with darker output and rotation to match the preview image

-I- Command CONVERT_ROTATE_MRW_CONTRP2DARK1: directory : "**DoContrPlus2Dark1**/" - performs raw conversion with much more contrasty and darker output, and rotation to match the preview image

Names of the directories should be (and are) self-explanatory. For example, “DoBrightP1” stands for increase-brightness-by-one-step; here is how you can verify this:

run_describe_command CONVERT_ROTATE_MRW_BRIGHTP2

-I- Command CONVERT_ROTATE_MRW_BRIGHTP1: directory - **DoBrightP1/** - performs raw conversion with a little brighter output and rotation to match the preview image

Note that the default command has two associated directories – one default and one explicit; in my configuration you need to put some preview image into “DoDefault/” directory to cause creation of a corresponding default preview.

The user is expected to analyze each preview and decide whether it requires a color correction and which one. Here are the choices:

a) the preview looks just right – you want the final image exactly like it

move the preview into KeepAsIs/ sorting directory

b) it's enough to look at the preview to conclude that this shot is hopeless

delete the preview

c) according to the preview, the image is so-so; you want to keep it “for a record” but don't want to waste time on it

move the preview into KeepAsIs/ sorting directory

d) the preview may benefit from one of the predefined color corrections configured in your LazyConv installation

move the preview into sorting directory linked to this color correction

e) you would like to try several predefined color corrections configured in your LazyConv installation

copy the preview into all the corresponding sorting directories

don't worry about leaving the preview in the default output directory too – it has no impact there

f) the preview is too off; you'd like to convert this image manually

move the preview into Manual/ sorting directory

g) you would like to make a huge poster out of this image, so you choose manual correction to achieve maximal quality

move the preview into Manual/ sorting directory

h) you want to have both final image made after this preview, and manually converted image

copy the preview into Manual/ sorting directory

i) you don't yet know what to do with this image and prefer to postpone the decision

leave the preview where it is - in the default output directory

Important note: color correction to be applied depends only on the correction directory where you put the preview. This preview image itself could be a result of some other color correction, but it doesn't matter.

Then perform the **run_correction** command. After it you will probably have several previews

of some of the images, and you have to look through them, choose which one(s) you like most, and delete the rest. As is already said, you may prefer to keep several final conversions of the same input image – keep all corresponding previews then.

Handling of image orientation.

The two cameras I worked with (Dimage A2 and Dynax 5D) don't rotate vertical shots automatically, even though the sensor is there, and probably EXIF info contains orientation tag. The converter used by LazyConv could be incapable of reading the orientation tag, so the rotation should be handled manually.

The current version of LazyConv gives minimal treatment to the image orientation:

- you specify rotation angle for **all** the images that are known to be vertical; you do this in session output-configuration file out_config.tcl (see Session input/output settings)

  set DEFAULT_ROTATION_FOR_VERT_IMAGES_CW -90

  the example above is for images shot with right hand up; otherwise it should be 90
- you rotate each relevant preview image (by 90 degrees to either side) so it stands vertically as it should
- the consequent **run_correction** and/or **run_conversion** commands will detect that and make all images produced from appropriate input/raw file vertical - by rotating them by the angle you specified in DEFAULT_ROTATION_FOR_VERT_IMAGES_CW

I know, this will not work properly if you process in one session both images taken with right hand up and with left hand up. If LazyConv is found to be useful, I'll implement a better algorithm.

If your camera rotates raw images internally, set DEFAULT_ROTATION_FOR_VERT_IMAGES_CW to 0 (zero).

Using Irfanview for image viewing and sorting.

The current version of LazyConv is configured to use Irfanview as image viewer. You can invoke Irfanview either directly from Windows (in any standard way), or through LazyConv's viewing commands.

Important general notes about Irfanview:

- to switch between window and full-screen mode, press <Enter>
- stick to full-screen mode whenever possible – it provides best image quality
- there's only one step of UNDO
- configuration is stored in a settings' file
  - if you start Irfanview in a standard way (like double-click on associated file or from command line), you are using default Irfanview settings file
  - if you start Irfanview through LazyConv's **run_browse_previews_in_dir** command, local (per directory) copy of settings' file is created for your

session

- some image-manipulation commands won't work in full-screen mode – color adjustment, crop, etc.
- when browsing images in a directory, Irfanview picks only those that existed at the time it was started; all the newer images are ignored

The LazyConv's viewing commands are:

1. **run_browse_previews_in_dir <full_or_relative_directory_path>**
    - opens in the viewer the first image in specified directory; you can then browse forward and backward by means of the viewer's native features
    - the single most useful viewing command
    - the name could be misleading – usable for both previews and final images
    - the most frequent use is:
        - run_browse_previews_in_dir ***Out*** - to see the ***previews***
        - run_browse_previews_in_dir ***Out/Final*** - to see the ***final conversions***
2. **run_show_all**
    - runs slide-show with all previews
    - not useful
3. **run_show_corrections**
    - runs slide-show with images that have more than one preview
    - intended for seeing only those images for which you have to make a choice
4. **run_show_last_corrections**

    runs slide-show with images that passed correction in the latest session

    provide fast access to the images you dealt with last; useful when you work on a large group of images

    there's a bug; the first image shown could sometimes be unrelated

Irfanview has a limitation – its image-sorting features (described below) are not available in slide-show mode. As a result, all LazyConv's run_show_* commands are not very useful. Just use **run_browse_previews_in_dir** command instead.

The following Irfanview image-viewing features are especially useful for work in LazyConv:

1. browsing all images in a directory
    - the sorting order is by filename only
    - press <Space> or <right-arrow> to go forth

- press <BackSpace> or <left-arrow> to go back

2. copy or move current image into a choice of 10 directories (“image-sorting”)
    - when you are viewing an image, press F7/F8 (menu: File->MoveFile / File->CopyFile) to access move/copy dialog
    - the Move_file(s) / Copy_file(s) dialog will appear that enables to choose one of 10 directories (linked to 0-9 keys) to put the file into
    - after you set a path, press corresponding 0-9 button to perform the actual copy/move
    - after you set and used a path, it will become available next time you invoke this dialog
    - if you start Irfanview by **run_browse_previews_in_dir** command, the initial path for all the 10 options will be your default output directory, so that you have less directory levels to browse
3. image rotation by 90 degrees
    - mostly for rotating vertical images
    - applied to JPEG previews in LazyConv; use lossless-JPEG-rotation feature of Irfanview
    - when viewing an image, press <Shift-J> (menu: Options->JPG-Lossless-Operations)
        - choose the required angle in the dialog box with up/down arrows and press <Enter>
        - the image is rotated in-place – your original (the preview) gets overridden
        - next time you invoke this feature, rotation angle you chose will be the default
4. fine image rotation and cropping

    you may use both of them for final converted images

    rotation: press <Ctrl-U> (menu: Image->Custom/Fine-Rotation) and follow instructions

    crop: choose the area you want with mouse, then press <Ctrl-Y>

        note, crop won't work in full-screen mode
5. deleting images

    press <Del> to delete the currently viewed image (moves it into Recycle Bin)

And don't forget to read through “Using ... Speech Recognition” for another interesting suggestion :).

Using "tkcon" TCL console.

As described in "Starting TCL interpreter", to start tkcon console, run TclInterp.exe from MS-DOS prompt or "Run" menu, or shortcut, while providing it with the path to the console TCL code (tkcon.tcl).

The following features of tkcon made it preferable to me:

- Command history
    - pressing <Up> and <Dn> at console prompt brings you previously executed command lines; moreover, they are stored when you close the console and available in future sessions
    - pressing <Up> at console prompt while the beginning of the command is already typed, brings you previously executed command lines that begin with the same prefix
    - you can edit a previously executed command line freely to make a new command out of it
- Path (Unix style) / command name expansion
    - if you press <Tab> after typing beginning of the command, tkcon suggests the ending:
        - if there's a single command with such prefix, its name is completed in your command line
        - if there are several commands with such prefix, their names are printed, but your command line isn't changed
- Unix-like and Windows-like key bindings; see tkcon.sourceforge.net/docs/bindings.html

The full tkcon documentation could be found at tkcon.sourceforge.net/docs/index.html .

Caveats:

1. The console screen is not refreshed while long commands are running.
2. You cannot break the execution of a command.

    though I'm checking this issue and may eventually find a solution

    if you really need to stop a long-running LazyConv command, kill the console window
3. The console sometimes stays unresponsive even after the command is finished.

    switch to other application, then back to the console; <Alt-Tab> works best in Windows

Changing console font for Windows.

I preset the console font to Courier 14, which is great for my 17-inch 1280*1024 LCD screen.

If you want to change the font size, edit TCL\Console\tkcon.tcl source file:

find all the occurrences of the word "Courier" in the file, and change the size

appropriately

you may want different sizes for the main window area and title-bar (as I do)

Using ... Speech Recognition (Windows Vista).

I eventually realized that most of the work with LazyConv is doable through Speech Recognition (built into Windows Vista). E.g. one can control RAW processing by voice. Since then it became my way of working with LazyConv.

Why this is desirable.

It's all about convenience. With voice commands one doesn't have to reach keyboard/mouse most of the time; this enables more comfortable sitting or even lying :). Going into extremes, imaging having a calibrated projector for the task... Eventually RAW conversion becomes enjoyable.

Why this is possible.

- LazyConv has a limited set of commands
- most of the interaction with LazyConv occurs through Irfanview, and the latter is Speech Recognition friendly
- console commands could still be typed in a regular way; one doesn't do it too much
- most of Windows operation obeys to Speech Recognition commands.

What are the drawbacks.

- requires Windows Vista (3-rd party Speech Recognition tools were not tested)
- the commanding goes a little slower than with mouse/keyboard

  (though I believe newer quad-core processors can resolve this)
- requires silent working environment

Please refer to Windows Vista documentation on how to set-up Speech Recognition and use it for general tasks.

Voice commands for Irfanview.

In most case you just say what you otherwise would press or click. The table below summarizes the most useful commands. Whatever shown in () are comments.

Action	Voice command
Move to the next image	Space
Move to the previous image	Backspace
Move to the first image	Home
Move to the last image	End
Switch to/from full-screen mode	Enter
Enlarge to 100%	Press Control H

Action	Voice command
Scroll left (in 100% view)	Home
Scroll right (in 100% view)	End
Scroll up (in 100% view)	Page Up
Scroll down (in 100% view)	Page Down
Start “Copy file” dialog	Press F8
Start “Move file” dialog	Press F7
In copy/move dialog: to select correction directory for 0..9 cell use keyboard and/or mouse	
Choose correction directory allocated to a 0..9 cell	(say number of the cell, then OK)
Rotate the JPEG image (in multiples of 90-degree)	Press Shift J (then follow the dialog)
Delete the image	Delete (then say Yes)
Maximize the window	Maximize that
Exit Irfanview	Escape (maybe twice)
Exit Irfanview	Close That

Voice commands for TCL console.

This is not a must and applicable for “tkcon” console only.

Here one may use speech Recognition to utilize “history” feature of “tkcon” TCL console.

Say “Press Up Arrow” to put the previously executed command at the console prompt.

Repeat saying “Press Up Arrow” to go several commands back.

Say “Enter” to execute the command staying at the prompt.

Overriding white balance.

When working with RAW images from a digital camera, color balance is defined by 3 or 4 numbers, depending on the kind of sensor – red, green and blue multipliers. Normally these numbers correspond to “color temperature” of the light source at the time the shot is taken. I.e. one set of numbers is used for sun, another one for fluorescent lamp, and so on. You may want to have numerous sets to fine-tune image colors.

The camera stores these numbers in the RAW file as a part of metadata.

In most cases the user doesn't override white balance at the time of conversion, since one of the following conditions holds:

- Auto white-balance was used and handled the scene appropriately
- The photographer employed correct white-balance preset in the camera
- The photographer set white-balance manually in the camera to achieve some desired effect.

But in some cases one of the above may go wrong, and conversion-time white-balance

override comes handy. A very common example: if part of the frame is sun-lit, and another part is in shadow, and the camera picks white-balance for sun light, the shadowy part appears distractingly bluish; I'd prefer to use shade-type white-balance; the sunny part will become reddish; not perfect but tolerable.

So, to override white balance for specific image you have to:

1. Decide which white balance you'd like to apply for this image
2. Obtain those 3-4 numbers that define white balance you chose
3. Tell LazyConv that specific image should be converted with your white-balance numbers instead of those recorded in the RAW file.

Obtaining white-balance numbers for camera presets.

Most of practical cases of white-balance errors could be resolved by knowing preset numbers for common lighting conditions. These sets of numbers are specific to ***individual camera sample***, and this section explains how to retrieve them.

1. Take a serious of shots in RAW mode (you can leave the lens cap on) with all the white-balance presets you are interested in – like “Sunny”, “Cloudy”, Tungsten”, etc. Keep track of the order in which you select the presets.
2. Take the RAW files to the computer – place them in one directory.
3. Read white-balance coefficients from the RAW files.

   The example below shows how to do it with dcraw from MS-DOS command prompt. You should change everything shown in ***bold-italic*** below to match your directory and file names.

   Copy-paste the below script into text editor like Wordpad, Notepad, or any code-oriented editor (not MSWord, StarOffice Writer or alike!), make the changes, then copy-paste into MS-DOS command prompt window.

   - update the directory path where you copied your RAW files
   - update the path of dcraw to match your system
   - update the name-pattern to match your RAW files.

   ```
   cd  /d <directory where you copied your RAW files>
   set DCRAW=”C:\ANY\LazyConv\DCRAW\dcraw.exe”
   for %f in (*.MRW) do @(
       echo %f
       %DCRAW% -v -i %f |find "Camera multipliers"
   )
   ```

   You are going to see output like this:

   ```
   PICT1080.MRW
   ```

Camera multipliers: 526.000000 256.000000 348.000000 256.000000

PICT1889.MRW

Camera multipliers: 535.000000 256.000000 355.000000 256.000000

PICT1998.MRW

Camera multipliers: 518.000000 256.000000 420.000000 256.000000

PICT2011.MRW

Camera multipliers: 525.000000 256.000000 359.000000 256.000000

The multipliers come in the following order: red, green-1, blue, green-2.

Note that in each set green-1 and green-2 are equal; this holds for most of the current sensors – except Fuji's super-CCD.

Now you should match between the RAW files above and the names of the white-balance presets in your camera. Then, either provide the presets manually (see "Providing white-balance overrides to LazyConv"), or use facilities of "camera_data" package (see "Built-in per-camera presets in LazyConv").

Providing white-balance overrides to LazyConv.

If a file named "***lazyconv_wb_ovrd.txt***" appears in a *root sort directory*, LazyConv takes white-balance numbers from this file for conversions/corrections triggered by previews found under this *root sort directory*. White-balance numbers from the RAW files themselves are ignored then.

Two examples below explain what is it "*root sort directory*".

1. work area created by a command like

   run_setup_work_area -origin <origin> -inp_dirs_root <input_dir> \

   -work_dirs_root <work_dir>

   has one *root sort directory* - **<work_dir>/Out**

2. work area created by a command like

   run_setup_work_area -origin <origin> -inp_dirs_root <input_dir> \

   -work_dirs_root <work_dir> -sort_dir_names {Group1 Group2 Group3}

   has three *root sort directories* - **<work_dir>/Out/Group1, <work_dir>/Out/Group2, <work_dir>/Out/Group3**

Format of "***lazyconv_wb_ovrd.txt***" files:

any comment – a line starting with "#" is ignored

<red-multiplier> <green1-multiplier> <blue-multiplier> <green2-multiplier>

Example:

white-balance coefficients (r g1 b g2) for {SAMPLE_MINOLTA_A2 Cloudy}

589.0 256.0 332.0 256.0

So, after you have a preview image created with whatever white-balance numbers, you can put it under sorting directory with white-balance override file, and from now on this preview will trigger conversions with the new white balance.

Note that only one white-balance setting per a RAW file *at a time* is supported in LazyConv. See “White balance is a property of RAW image” for more on this.

Built-in per-camera presets in LazyConv.

The “camera_data” package in LazyConv enables convenient storage and access of white-balance presets for any number of specific camera samples.

Storing the presets.

LazyConv ships with several sample white-balance files – for the cameras the author played with.

Note that white-balance numbers may vary for different samples of the same camera model.

To create white-balance file for your camera, make a copy of any existing file, rename it appropriately, and replace camera name and the numbers by yours. You may add, rename or delete presets if you need. Be sure that you save the file as a regular text, and its extension is “.tcl”.

If you installed LazyConv as in example in “Installation of LazyConv”, the white-balance files are in directory C:\ANY\LazyConv\TCL\Work\camera_data .

Example.

Below is a relevant extract from a sample white-balance file; fragments that require changes are shown in **bold**:

```
proc ::camera_data::define_camera__SAMPLE_SONY_ALPHA_100 {} {
   set cName "SAMPLE_SONY_ALPHA_100"
   add_camera_id $cName

   add_camera_wb_preset $cName "Sunny"       {467.0 256.0 446.0 256.0}
   add_camera_wb_preset $cName "Shade"       {554.0 256.0 389.0 256.0}
   add_camera_wb_preset $cName "Cloudy"      {497.0 256.0 394.0 256.0}
   add_camera_wb_preset $cName "Tungsten"    {294.0 256.0 962.0 256.0}
   add_camera_wb_preset $cName "Fluorescent" {463.0 256.0 749.0 256.0}
   add_camera_wb_preset $cName "Flash"       {498.0 256.0 422.0 256.0}

   print_defined_wb_presets $cName
```

```
}
# register this camera at the time of loading the code
::camera_data::define_camera__SAMPLE_SONY_ALPHA_100
```

Accessing the presets.

In order to learn what presets are available, type:

- **camera_data::print_defined_wb_presets** to get all available presets
- **camera_data::print_defined_wb_presets** <camera_name> to get presets for specified camera

Example (in TCL console):

MyRun> ***camera_data::print_defined_wb_presets*** SAMPLE_MINOLTA_A2

```
SAMPLE_MINOLTA_A2  : ----SAMPLE_MINOLTA_A2----Tungsten----    : 282.0 256.0 751.0 256.0
SAMPLE_MINOLTA_A2  : ----SAMPLE_MINOLTA_A2----Fluorescent----: 456.0 256.0 553.0 256.0
SAMPLE_MINOLTA_A2  : ----SAMPLE_MINOLTA_A2----Sunny----       : 525.0 256.0 359.0 256.0
SAMPLE_MINOLTA_A2  : ----SAMPLE_MINOLTA_A2----Shade----       : 633.0 256.0 324.0 256.0
SAMPLE_MINOLTA_A2  : ----SAMPLE_MINOLTA_A2----Flash----       : 546.0 256.0 341.0 256.0
SAMPLE_MINOLTA_A2  : ----SAMPLE_MINOLTA_A2----Cloudy----      : 589.0 256.0 332.0 256.0
```

As described in “Providing white-balance overrides to LazyConv”, for LazyConv to override white balance of an image, the user has to:

1. Put appropriate “***lazyconv_wb_ovrd.txt***” file in a root sort directory.
   - performed by the following LazyConv command:

   camera_data::write_wb_ovrd_file_in_dir <camera_name> <preset_name> <directory>

   Example that creates white-balance override file in the current directory (.):

   camera_data::write_wb_ovrd_file_in_dir SAMPLE_MINOLTA_A2 Cloudy .
2. Place the preview **under** this directory.
   - Place it into this root sort directory to obtain the same color correction.
   - Place it into a correction directory under this root sort directory to obtain a different color correction.
3. Run **run_remake_wb_ovrd_previews** command.
   - If the original preview was in the root sort directory, it is replaced by the new preview (the same name) with specified white-balance but the same color correction as the original one.
   - If the original preview was in the correction directory, it is preserved; the new preview has a different name (defined by the color correction); be sure to delete the original preview before the final conversion.

- The **run_remake_wb_ovrd_previews** command could be run either for one explicitly-given root sort directory (syntax: **run_remake_wb_ovrd_previews** <directory_name>), or for all root sort directories in the work area (syntax: **run_remake_wb_ovrd_previews**)
- Note 1: **run_remake_wb_ovrd_previews** issues **run_correction** command as a part of its flow; as a result all the corrections requested under the work area will take place, not only those relevant to white-balance overrides. This shouldn't be a problem usually; just be aware.
- Note 2: **run_remake_wb_ovrd_previews** without explicit root sort directory will recreate the previews it finds in the root directory for auto white-balance. This is desirable if the user wanted to change from preset white-balance to auto white-balance. But this is waste of time if those previews are already made with auto white-balance. Again, the user can easily avoid this scenario.

White balance is a property of RAW image.

White-balance setting is naturally a part of the captured image itself – nothing more than a way for the software to interpret image data. It is detected at the time of capture, and in most cases stored in the image's metadata.

In case the white balance needs to be overridden, the most obvious approach would be to alter white-balance coefficients in the RAW file itself. While this would ensure the further flow is straightforward and fool-proof, there are other considerations too.

LazyConv is designed not to change input (RAW) files, even when it comes at a price, and white-balance override is the case. The author aimed at getting the alternative white-balance override solution as elegant as possible, but ... read the text above.

Usage strategies for white-balance overrides.

This section presents several recommended strategies of dealing with white-balance overrides.

1. Create separate additional work area for each white-balance preset; input (RAW) directory is shared with the “main” work area (where you ran “run_preview_all”).
    - **run_process_unlisted** command becomes unusable (dangerous), since no single work area keeps track of all input (RAW) files altogether
    - you need to work in several TCL consoles (per work area) or switch work area each time
2. Create separate additional work area for each white-balance preset with its own copy of the input (RAW) directory.
    - takes more space, since input (RAW) files are duplicated
    - **run_process_unlisted** command should be separately run in each work area

- you need to work in several TCL consoles (per work area) or switch work area each time

3. Create per-preset sorting directories in the single work area.
    - this approach is preferred by the author
    - note, you usually need a directory for camera/auto white balance too; this directory has no white-balance override file
    - may collide with a need to use multiple-sorting-directories feature for another purpose
    - the user should ensure that previews of the same image with different white balance don't coexist; this scenario is not supported by LazyConv, and it cannot always detect the problem
4. (A suggestion; not yet tested.) Create separate additional work area for each white-balance preset; input (RAW) directory is shared with the “main” work area through individual soft links to each one of the RAW files.
    - you need to work in several TCL consoles (per work area) or switch work area each time
    - requires Windows Vista (or newer).

If you choose the approach of per-preset sorting directories in the single work area, you can create directories for all registered presets of your camera by providing “**-wb_ovrd_dirs_for_camera <camera_name>**” switch to run_setup_work_area command.

Example:

> **run_setup_work_area** -origin MINOLTA -inp_dirs_root C:/ANY/Test_LazyConv/Raw/ -work_dirs_root C:/ANY/Test_LazyConv/MyRun/ **-wb_ovrd_dirs_for_camera SAMPLE_MINOLTA_A2** -clean yes

- Be sure not to run this in a work area that already has a different directory structure (single sorting directory or directories for another camera); this won't work even with “-clean yes”.
- The directories with white-balance overrides are named after the preset names; the directory without the override is named “AutoWB”.

After you have one or more directories with valid white-balance override files, the recommended work flow is as follows:

1. **run_preview_all** in the main work-area (no white-balance override); previews with camera-set (or automatic) white-balance are created
2. look at the default previews, distribute them between per-preset sorting directories; images that don't require white-balance override should be put into a directory without white-balance override file
    - if you wish to compare default previews against those with white-balance overrides, keep the original previews in-place (where created by run_preview_all)- i.e. file-copy them instead of file-move; don't forget to

delete one of the versions afterwards

3. **run_remake_wb_ovrd_previews** in all work areas that have one or more directories with white-balance override files
    - if you preserved the original previews, compare them to the new versions and delete whatever you don't need; LazyConv cannot handle conflicting white-balance settings
4. continue usual LazyConv work-flows in all sorting directories or work-areas you have
    - **run_browse_previews_in_dir** always works in one explicitly given directory
    - for multiple sorting directories **run_correction** serves all of them at once; each new preview is saved under the sorting directory of the preview that triggered its creation
    - for multiple sorting directories **run_conversion** serves all of them at once; all the final images are created in the common directory under the root output directory – as if there were no multiple sorting directories
    - for multiple work areas you should run all correction/conversion commands individually
    - **run_process_unlisted** could be used as usual for multiple sorting directories but not for multiple work areas
5. ?TODO?

The author's recommendation is to follow the approach of per-preset sorting directories in the single work area.

See “Basic white-balance overrides”, “Using sorting directories with white-balance overrides” and “Mainstream usage of white-balance overrides” for demos on working with white-balance overrides.

Common pitfalls with white-balance overrides.

This section refers to the “per-preset sorting directories in the single work area” strategy.

One may run into several pitfalls when working with white-balance overrides (presets). Most of them happen when conflicting previews with conflicting white-balance settings are allowed to exist in the same work area (see “White balance is a property of RAW image”). This scenario should be avoided; in case the user needs to compare between different white-balance settings, one of the image version should be moved outside of the current LazyConv work area.

LazyConv will detect the conflict in case one command runs into the need to process several previews of the same input image with different white-balance settings. The process is stopped at the first conflict being encountered; the work-area is left in an intermediate state requiring manual cleanup.

The table below shows some of the scenarios that the user may encounter; most of them

associated with the above mentioned conflicts. Don't get frightened though; just avoid these conflicts, and LazyConv will work smoothly.

N#	The case	The outcome	Error message(s)
1	*run_remake_wb_ovrd_previews* <*wb_ovrd_dir*> issued while same preview stayed both in <wb_ovrd_dir> and in directory for auto WB	Previews in <wb_ovrd_dir> replaced with new versions; previews in directory for auto WB not affected	None
2	*run_remake_wb_ovrd_previews* issued while same preview stayed both in two different directories with WB overrides	The process stopped at the time conflict is detected; work-area requires manual cleanup	-E- Conflicting white-balance overrides for ...: ... the latter from preview image ...
3	*run_remake_wb_ovrd_previews* issued while same preview stayed both in <wb_ovrd_dir> and in directory for auto WB	The process stopped at the time conflict is detected; work-area requires manual cleanup	-E- Conflicting white-balance overrides for ...: ... the latter from preview image ...
4	*run_correction* issued while same preview stayed in **correction directories** both under one directory with WB override and under directory for auto WB	The process stopped at the time conflict is detected; work-area requires manual cleanup	-E- Conflicting white-balance overrides for ...: ... the latter from preview image ...
5	*run_conversion* issued while differently corrected previews of the same image stayed under two different directories with WB overrides	The process stopped at the time conflict is detected; work-area requires manual cleanup	-E- Conflicting white-balance overrides for ...: ... the latter from preview image ...
6	*run_remake_wb_ovrd_previews* issued twice – i.e. a directory with WB override already had WB-corrected previews when *run_remake_wb_ovrd_previews* is run for the second time	(Identical) previews with WB override are re-created.	None
7	*run_remake_wb_ovrd_previews* issued while the work area has both previews intended for WB override AND previews intended for “regular” correction	All the actions occur – remake for WB override and “regular” correction, even though the latter wasn't explicitly requested	None
8	*run_remake_wb_ovrd_previews* issued while some previews (not requiring WB override) stayed in the root directory for auto WB	These previews are recreated with auto WB. The resulted files are equal to the original.	None

Configuring LazyConv.

Important note: whenever you make changes in LazyConv's TCL source files, you should open a new TCL console for the changes to take effect.

Adding or fixing raw-file origins (supported cameras).

Origin definitions are located in TCL/Work/rawconvert/origin.tcl and are pretty self-explanatory.

For example, names of Minolta raw files look like “PICT1234.MRW”, and their origin definition in LazyConv is:

```
.....
        "MINOLTA" {
           set ext "MRW"
           set fullPattern {{PICT[0-9]+\.MRW}}
           set convertCmd  CONVERT_ROTATE_MRW_DEFAULT
        } \
.....
```

Note that <ext> and <fullPattern> are case-sensitive. <fullPattern> must follow TCL regular expression syntax.

If definition for your origin is incorrect, change whatever required in the **bold** part.

If definition for your origin is missing, copy-paste any existing definition and change the **bold** part appropriately.

Output specifications.

LazyConv uses *output specification* (“outspec”) to define how and where the output image is to be stored.

- The “how” part means image format together with optional compression settings.
- The “where” part means output directory and filename.

In the end of the day, *output specification* is a TCL array with the following fields (sorry for field names being so long):

$imageproc::outspecName	<- pure name for the output image file
$imageproc::outspecDir	<- directory for the output image file
$imageproc::outspecFormat	<- format (standard) for the output image file
$imageproc::outspecCompress	<- compression/quality for the output image file
$imageproc::outspecScale	<- not supported
$imageproc::outspecBitDepth	<- bits per color in the output image file

$imageproc::outspecSrcName <- full path of the input file

Usually the user specifies the “how” part of the *output specification*, the rest is completed by the LazyConv at the conversion time. Thus, the user should only set output format (JPEG, etc.), compression level and bit depth.

To simplify the task, LazyConv employs *outspec templates* – predefined partial *output specifications* that define everything in the “how” part. So, instead of setting each “how”-related parameter separately, you choose the *outspec template* that suits your needs best. How to do it is explained in “Session input/output settings” - in the section devoted to the output configuration file.

LazyConv is shipped with the following preconfigured *outspec templates*; I believe this is enough for any usage scenario:

- TIFF8 - 8-bit TIFF with lossless LZW compression
- JPEGok - 8-bit JPEG with quality level 90
- JPEGlarge - 8-bit JPEG with quality level 95
- JPEGbest - 8-bit JPEG with quality level 100
- TIFF16 - 16-bit TIFF with lossless LZW compression
    - as an ultimate output supported only by “DCRAW based “real” raw conversion commands”

The preconfigured *outspec templates* are located in TCL/Work/imageproc/outspec.tcl; perform the changes there if you wish to fine-tune the parameters.

Names of preconfigured *outspec templates* are case-sensitive.

Session input/output settings.

Inputs' and outputs' configurations for a session are placed into inp_config.tcl and out_config.tcl files by the **run_setup_work_area** command; though you can create and/or edit these files manually. These files contain valid TCL language statements.

Here is MyRun\inp_config.tcl that would be created by the example in “Setting-up LazyConv for a session”

```
# inp_config.tcl - generated by make_input_config_file
#############################################################################
set RAW_EXT  MRW
set RAW_NAME_PATTERN_WITH_EXT  {PICT[0-9]+\.MRW}
set RAW_IMG_ROOT   C:/ANY/LazyConv/tcl/Work/Run/Raw/

set PREVIEW_EXT_LIST     [list JPG TIF]
set SORTED_IMG_DIRS      [split "C:/ANY/LazyConv/MyRun/Out" ", \t"]

# Default conversion command is an optional setting
set CONVERT_CMD_NICKNAME       CONVERT_ROTATE_MRW_DEFAULT

# command applied to input files without previews
```

```
set PROCESS_UNLISTED_CMD       RENAME_UNLISTED_FILE
#############################################################################
```

Comments:

Input-file extension (RAW_EXT) is set to MRW to match that of Minolta raw files.

Input-file full name pattern (RAW_NAME_PATTERN_WITH_EXT) is a regular expression to match Minolta raw files. Note, it is **case-sensitive**.

Root directory **under** which to look for input files is defined in RAW_IMG_ROOT. Inputs could be placed either directly there, or in any its sub-directory.

Extensions of preview files (PREVIEW_EXT_LIST) are taken as TCL list; this is to support previews in various formats.

You can specify (in SORTED_IMG_DIRS) one or more set(s) of image-sorting directories – where previews are created and looked for. The example of LazyConv session with two sets of image-sorting directories is provided in “Using two groups of image-sorting directories”.

The syntax of SORTED_IMG_DIRS may look confusing; if you create it manually, here is a more straightforward notation:

```
set SORTED_IMG_DIRS      [list "C:/ANY/LazyConv/MyRun/Out"]
```

Conversion command to be used by default – by **run_preview_all** and for conversion without any color correction, is set by CONVERT_CMD_NICKNAME.

The command to be applied to input files without previews by **run_process_unlisted** is set by PROCESS_UNLISTED_CMD; if you want to change it, you should first define a relevant replacement command for LazyConv; not recommended.

Here is MyRun\out_config.tcl that would be created by the example in “Setting-up LazyConv for a session”

```
# out_config.tcl - generated by make_output_config_file
#############################################################################

set PREVIEW_OUTSPEC_TEMPLATE_NAME    JPEGok
set OUTSPEC_TEMPLATE_NAME      TIFF8
set DESTINATION_DIR      C:/ANY/LazyConv/MyRun/Out
set DEFAULT_ROTATION_FOR_VERT_IMAGES_CW    -90
set TMPFILE_PURENAME     converted
#############################################################################
```

Comments:

Setting PREVIEW_OUTSPEC_TEMPLATE_NAME defines output format for preview images; see “Output specifications”.

Setting OUTSPEC_TEMPLATE_NAME defines output format for final images; see “Output specifications”.

Setting DESTINATION_DIR tells the default output directory – where **run_preview_all** writes its outputs.

Setting DEFAULT_ROTATION_FOR_VERT_IMAGES_CW tells how to rotate outputs of **run_correction** and **run_conversion** if the corresponding preview image is vertical;

see “Handling of image orientation”.

The TMPFILE_PURENAME defines pure file-name for converter's intermediate files.

Presets for Irfanview.

This is a more convenient alternative to customization made through Options->Properties menu in Irfanview. While some settings are common, others are coupled to the LazyConv session; presets mechanism enables to prepare LazyConv-specific or session-specific Irfanview configuration on-the-fly.

Irfanview customization settings are stored in **i_view32.ini** file (“INI file” further); you may keep several such files on the computer – in different directories. One is default – it's located in the same directory with Irfanview executable. If you specify directory with INI file explicitly when starting Irfanview (by */ini* switch), it will be used instead of the default.

Format of Irfanview initialization file.

The format of **i_view32.ini** file is as follows:

The file is split into logical sections; each section starts from a line containing its name in brackets [] and ends where the next section starts.

Example:

```
[Viewing]
... settings for viewing ...
[TIFF]
...
```

Each setting is specified in a separate line as <setting_name>=<value>.

Example:

```
ShowFullScreenName=1
```

(the above setting from [Viewing] section tells to show image full path when viewed in full-screen mode)

Irfanview presets in LazyConv.

In LazyConv, presets for Irfanview are stored in TCL/Work/cnv_config/iview_presets.tcl .

The code is well commented, so it's simple to find and change the setting you want.

Currently there's no simple way to add a new setting through LazyConv; I may consider making such enhancement.

Defining conversion commands.

(This chapter requires basic understanding of computer programming.)

Conversion commands differ depending on converter being used, but here is the common background.

In LazyConv the conversion is performed by invoking system-specific command line[s] to

operate on input (raw) files. LazyConv first builds these command lines given input files, conversion parameters, output format and location, and then executes them.

In general, conversion of each raw image is made by two consecutive actions (shown in **bold**):

1. (input/raw file) -> **raw conversion engine** -> (intermediate standard image)
2. (intermediate standard image) -> **postprocessing** -> (final standard image)

Standard image means regular TIFF/JPEG/BMP file. Postprocessing on the 2-nd stage could be any operation applicable to standard images – like change of brightness, contrast enhancement, complex curve application, etc.

Command definition includes specifying three aspects:

1. How to build command that invokes raw-conversion engine.
2. How to build command that invokes postprocessing.
3. What sorting directory to link to the command.

Each command is assigned an unique *nickname* used to identify it.

A wrapper for particular converter provides *command-building procedure* that takes the following textual arguments:

- input/raw file

  full path
- conversion settings in converter–specific form

  for Irfanview - directory with Irfanview configuration file

  for dcraw – a set of dcraw command line switches
- a rule to build the output file name – a suffix to append to the input file name

  if the suffix is “_ivrdef”, output name for “pict012.mrw” raw file will be “pict0012_ivrdef.tif” of “pict0012_ivrdef.jpg” depending on the output format
- output format and directory (see “Output specifications” chapter)
- name of TCL procedure that performs postprocessing

  this procedure both builds and executes operating-system-specific command line

  see “Postprocessing procedures for LazyConv” for the syntax details

The *command-building procedure* returns a valid TCL command (in the form of list) that should perform the required conversion.

When defining a command, you specify the chosen *command-building procedure* and provide its arguments as “-<argument_name> <argument_value>”; the examples are shown below.

Some of the arguments (input and output paths, etc.) only become known at the time of execution. LazyConv employs *argument-placeholders* to provide a place where to insert the specific value during the execution:

- **@iName@** for input file paths
- **@outSpec@** for output specification
- **@prName@** for preview image path

For example, if command definition includes ***-inpFile "@iName@"***, and the input file is “E:/MyDir/myfile.tif”, *command-building procedure* will obtain arguments “-inpFile E:/MyDir/myfile.tif”.

LazyConv provides two TCL procedures for command definition – **xx::define_cmd** for defining a command “from scratch” and **xx::define_modified_cmd** for defining a new command by partially modifying an existing (“base”) one.

xx::define_cmd <command nickname> **[list** \

-Description "<textual command description>" \

-BuildProc "<name of command building procedure>" \

-ONameSuffix "<suffix to build output filename>" \

-inpFile "input file path or @iName@ placeholder" \

-outspecArrName "output specification or @outSpec@ placeholder"

-<name of argument> "<value for argument>" \

... ... \

-<name of argument> "<value for argument>" **]**

Command nickname, description, name of command-building procedure, input file path and output specification are mandatory, the rest of the arguments could be specific for command-building procedure.

xx::define_modified_cmd <new command nickname> \

<base command nickname> **[list** \

-Description "<textual command description>" \

-<name of overridden argument> "<new value for argument>" \

-<name of overridden argument> "<new value for argument>" \

... ... \

-<name of overridden argument> "<new value for argument>" **]**

New and base commands' nicknames, as well as description, are mandatory. Then, only the arguments being changed are specified. The same goes for command-building procedure.

In order to make the new command available in the conversion sessions, you should link it to some sorting/correction directory. Then, when a preview is put into this directory, the next invocation of **run_correction** will cause creating preview image with the new command.

The following call should occur to perform the command-to-directory linkage:

xx::map_reldir_to_cmd "<directory name>" <command nickname>

The best place for this call is together with the similar calls for the standardly shipped commands. Look for, say, CONVERT_ROTATE_MRW_CONTRP1 in the LazyConv code, and insert similar lines.

Examples of command definition are shown in the following converter-specific subsections.

Irfanview based raw conversion commands.

Irfanview wrapper belongs to “snapshot” type (see “Choosing the right conversion scheme”).

Irfanview wrapper has two command-building procedures that are similar except that one of them supports image orientation (see “Handling of image orientation”):

- **::irfanview::build_cmd__default_convert_mrw** - without image-orientation support
- **::irfanview::build_cmd__default_convert_rotate_mrw** - with image-orientation support

Here is the definition of the Irfanview-based default raw conversion command as appears in TCL/Work/tool_wrap_main/wrap_rconv_iview.tcl:

xx::define_cmd CONVERT_ROTATE_MRW_DEFAULT **[list** \

-Description "performs raw conversion with default colors and rotation to match the preview image" \

-BuildProc "::irfanview::build_cmd__default_convert_rotate_mrw" \

-ONameSuffix "_ivrdef" **-inpFile** "@iName@" **-outspecArrName** "@outSpec@" \

-iniDir "$INIDIR_DEFAULT" **-previewPath** "@prName@" **-postConvProc** ""**]**

All the arguments are mandatory; argument not used in specific conversion command should be explicitly passed as an empty string.

Definition of the similar command but without image-orientation support differs only in two arguments: name of command building procedure is “::irfanview::build_cmd__default_convert_mrw”, and **-previewPath** argument should not appear at all.

Raw conversion parameters are transferred through -**iniDir** argument that tells full path to **Irfanview settings directory**; the latter should contain file **i_view32.ini** with raw-conversion related settings (see “Format of Irfanview initialization file”).

The raw-conversion settings we need to specify, belong to [Plugins] section; here is the example for default conversion that I use for my two Minolta digicams; the comments are provided for explanation; they cannot be included into the actual settings' file:

```
[Plugins]
CRW_AWB=0         <- do not use auto white-balance
CRW_Bright=0.85   <- brightness
CRW_CWB=1         <- do use use camera white-balance (from EXIF data)
CRW_Gamma=0.60    <- gamma
```

Irfanview versions prior to 3.98 had another two very useful flags: blue- and red multipliers; they enabled to introduce constant color corrections for all the images; value less than 1.0 decreases the corresponding channel, while greater than 1.0 – increases it.

```
CRW_Blue=0.97     <- blue multiplier
CRW_Red=0.97      <- red multiplier
```

Irfanview used to ship with red and blue multipliers preset to 1.0; I changed both to 0.97 in order to correct consistent magenta cast I had on the images from Minolta A2 and Minolta 5D. You may find other settings that suit better your equipment and your personal taste. Though this possibility doesn't exist in the new versions anyway.

When you create a new Irfanview settings directory, it's recommended to clone a standard one and make the required changes.

To demonstrate defining new commands based on existing ones, here is the definition of the Irfanview-based raw conversion command with little contrast enhancement:

xx::define_modified_cmd CONVERT_ROTATE_MRW_CONTRP1DARK1 \

CONVERT_ROTATE_MRW_DEFAULT **[list** \

-Description "performs raw conversion with a little more contrasty and darker output, and rotation to match the preview image" \

-ONameSuffix "_ivrcnp1dk1" **-postConvProc** postconvert::contrP1dark1**]**

Note the mandatory change of the output filename suffix.

In the general case, you may want to modify both the raw-conversion parameters and postprocessing procedure to achieve the desired color correction.

Definitions of all the raw-conversion commands are located in procedure "**::irfanview::define_conversion_commands**" in TCL/Work/tool_wrap_main/wrap_rconv_iview.tcl . Put your new commands there too.

Linkage of conversion commands to sorting/correction directories occurs in procedure "**::irfanview::set_default_reldir_to_cmd_mapping**" in TCL/Work/tool_wrap_main/wrap_rconv_iview.tcl . Here is an example:

xx::map_reldir_to_cmd "DoBrightP1" CONVERT_ROTATE_MRW_BRIGHTP1

DCRAW based "snapshot" raw conversion commands.

From the user's perspective, defining "snapshot"-kind commands based on dcraw is very similar to how it's done with Irfanview, even though the engine's interface is different. Most of the difference is hidden in the command-building procedures; there are two of them, just like with Irfanview:

- **::dcraw::build_cmd__default_convert_raw** - without image-orientation support
- **::dcraw::build_cmd__default_convert_rotate_raw** - with image-orientation support

Here is the definition of the dcraw-based default "snapshot" raw conversion command as appears in TCL/Work/tool_wrap_main/wrap_rconv_dcraw.tcl:

xx::define_cmd CONVERT_ROTATE_MRW_DEFAULT **[list** \

-Description "performs raw conversion with default colors and rotation to match the preview image" \

-BuildProc "::dcraw::build_cmd__default_convert_rotate_raw" \

-ONameSuffix "_ivrdef" **-inpFile** "@iName@" **-outspecArrName** "@outSpec@" \

-wbOvrdCoefs "@wbOvrdCoefs@" **-colorFlags** "$COLORS_DEFAULT" \

-previewPath "@prName@" **-postConvProc** ""**]**

All the arguments are mandatory; argument not used in specific conversion command should be explicitly passed as an empty string.
Definition of the similar command but without image-orientation support differs only in two arguments: name of command building procedure is "::dcraw::build_cmd__default_convert_raw", and **-previewPath** argument should not appear at all.

Raw conversion parameters are transferred through -**colorFlags** argument; they exactly match dcraw command-line switches.

With the official version of dcraw only one flag is in use:

"**-b** <brightness>".

The private version of dcraw available at www.lazyconv.com has additional flag for specifying color-channel multipliers:

"**-R** <red_multiplier> <green_multiplier> <blue_multiplier> <green_multiplier>"

Color-channel multipliers introduce constant color corrections for all the images; value less than 1.0 decreases the corresponding channel, while greater than 1.0 – increases it.
dcraw wrapper holds sets of color-setting flags to be passed as "**-colorFlags**" argument in conversion command definitions.

For the official version of dcraw:

```
variable COLORS_DEFAULT  "-b 0.85"
variable COLORS_BRIGHTER "-b 1.00"
variable COLORS_DARKER1  "-b 0.55"
variable COLORS_DARKER2  "-b 0.50"
```

For the private version of dcraw with color-channel multipliers:

```
variable COLORS_DEFAULT  "-b 0.85 -R 0.95 1 0.95 1"
variable COLORS_BRIGHTER "-b 1.00 -R 0.95 1 0.95 1"
variable COLORS_DARKER1  "-b 0.55 -R 0.95 1 0.95 1"
variable COLORS_DARKER2  "-b 0.50 -R 0.95 1 0.95 1"
```

You should comment-out the set of flags you are not going to use – by typing "#" in the beginning of corresponding lines, and un-comment the other set. Lazyconv ships with the set of flags including channel multipliers being active. This means you should modify the code if you use official version of dcraw.

The "**-wbOvrdCoefs**" flag enables providing white-balance override as a set of four coefficients; see "Overriding white balance". This flag is controlled by LazyConv itself; don't change it in explicit command definitions.

To demonstrate defining new commands based on existing ones, here is the definition of the

dcraw-based raw conversion command with little contrast enhancement:

xx::define_modified_cmd CONVERT_ROTATE_MRW_CONTRP1DARK1 \

CONVERT_ROTATE_MRW_DEFAULT **[list** \

-Description "performs raw conversion with a little more contrasty and darker output, and rotation to match the preview image" \

-ONameSuffix "_ivrcnp1dk1" **-postConvProc** postconvert::contrP1dark1**]**

Note the mandatory change of the output filename suffix.

This specific definition is identical to its Irfanview-based sibling, because postprocessing mechanism for dcraw wrapper is the same as for Irfanview.

In the general case, you may want to modify both the raw-conversion parameters and postprocessing procedure to achieve the desired color correction.

Definitions of all the raw-conversion commands are located in procedure "**::dcraw::define_conversion_commands**" in TCL/Work/tool_wrap_main/wrap_rconv_dcraw.tcl . Put your new commands there too.

Linkage of conversion commands to sorting/correction directories occurs in procedure "**::dcraw::set_default_reldir_to_cmd_mapping**" in TCL/Work/tool_wrap_main/wrap_rconv_dcraw.tcl . Here is an example:

xx::map_reldir_to_cmd "DoBrightP1" CONVERT_ROTATE_MRW_BRIGHTP1

DCRAW based "real" raw conversion commands.

The "real" type dcraw-based wrapper is TCL/Work/tool_wrap_main/wrap_rconv_cam_dcraw.tcl . Its default command is designed to approximate in-camera JPEGs.

The two command-building procedures are:

- **::dcraw::build_cmd__default_convert_raw** - without image-orientation support
- **::dcraw::build_cmd__default_convert_rotate_raw** - with image-orientation support

While they are named the same as in "snapshot: type converter, they work differently.

Here is the definition of the dcraw-based default "real" raw conversion command as appears in TCL/Work/tool_wrap_main/wrap_rconv_cam_dcraw.tcl:

xx::define_cmd CONVERT_ROTATE_MRW_DEFAULT **[list** \

-Description "performs raw conversion with default colors and rotation to match the preview image" \

-BuildProc "::dcraw::build_cmd__default_convert_rotate_raw" \

-ONameSuffix "_cvrdef" **-inpFile** "@iName@" **-outspecArrName** "@outSpec@" \

-wbOvrdCoefs "@wbOvrdCoefs@" **-colorShift** "$COLOR_SHIFT" \

-colorFlags "$COLORS_DEFAULT" **-previewPath** "@prName@" \

-postConvProc ""]

All the arguments are mandatory; argument not used in specific conversion command should be explicitly passed as an empty string.
Definition of the similar command but without image-orientation support differs only in two arguments: name of command building procedure is "::dcraw::build_cmd__default_convert_raw", and **-previewPath** argument should not appear at all.

Raw conversion parameters are transferred through **-colorShift** and **-wbOvrdCoefs** arguments; they exactly match dcraw command-line switches.

With the official version of dcraw only one flag is in use (and mandatory) in "**-colorShift**", even though it has no action:

"**-b** 1.0".

The private version of dcraw available at www.lazyconv.com has additional flag for specifying color-channel multipliers:

"**-R** <red_multiplier> <green_multiplier> <blue_multiplier> <green_multiplier>"

This flag works the same as in "DCRAW based "snapshot" raw conversion commands".

The "**-wbOvrdCoefs**" flag enables providing white-balance override as a set of four coefficients; see "Overriding white balance". This flag is controlled by LazyConv itself; don't change it in explicit command definitions.

Unlike "DCRAW based "snapshot" raw conversion commands", lightness is controlled by "piping" dcraw output through Imagemagick "convert" command. The lightness correction is defined by "**-colorFlags**" flag; any color-correction parameters accepted by Imagemagick "convert" are syntactically valid here. LazyConv ships with the following predefined groups:

variable COLORS_DEFAULT "-gamma 2.2 -sigmoidal-contrast 6,40%"

variable COLORS_BRIGHTER1 "-gamma 2.4 -sigmoidal-contrast 6,40%"

variable COLORS_BRIGHTER2 "-gamma 2.6 -sigmoidal-contrast 6,40%"

variable COLORS_DARKER1 "-gamma 2.0 -sigmoidal-contrast 6,40%"

variable COLORS_DARKER2 "-gamma 1.6 -sigmoidal-contrast 6,40%"

The "**-postConvProc**" flag works in the same manner as in "DCRAW based "snapshot" raw conversion commands".

Just like in "DCRAW based "snapshot" raw conversion commands", it's possible to define new commands based on existing ones. Since the syntax is very similar, there's no need in additional examples.

Definitions of all the raw-conversion commands are located in procedure "**::dcraw::define_conversion_commands**" in TCL/Work/tool_wrap_main/wrap_rconv_cam_dcraw.tcl . Put your new commands there too.

Linkage of conversion commands to sorting/correction directories occurs in procedure "**::dcraw::set_default_reldir_to_cmd_mapping**" in TCL/Work/tool_wrap_main/wrap_rconv_cam_dcraw.tcl .

Conversion commands for standard images.

These are based solely on Imagemagick.

As a first attempt, I made the simplest possible implementation: the first stage (where raw-conversion engine used to be called) does virtually nothing; then postprocessing procedure applies the required correction. This enabled me to reuse the postprocessing procedures written for the raw conversion.

Just as with Irfanview and raw conversion, Imagemagick wrapper for standard images has two command-building procedures that are similar except that one of them supports image orientation (see "Handling of image orientation"):

- **::LazyConv::build_cmd__default_convert_img** - without image-orientation support
- **::LazyConv::build_cmd__default_convert_rotate_img** - with image-orientation support

Here is the definition of the Imagemagick-based default image conversion command as appears in TCL/Work/tool_wrap_main/wrap_iconv_im.tcl (though this specific command will have no effect other than file-format conversion):

xx::define_cmd CONVERT_ROTATE_IMG_DEFAULT **[list** \

-Description "performs image conversion with default colors and rotation to match the preview image" \

-BuildProc "::iconv::build_cmd__default_convert_rotate_img" \

-ONameSuffix "_imrdef" **-transCmdList** {} \

-inpFile "@iName@" **-outspecArrName** "@outSpec@" \

-previewPath "@prName@" **-postConvProc** ""**]**

All the arguments are mandatory; argument not used in specific conversion command should be explicitly passed as an empty string or empty list according to its type.

Arguments are similar to that of the Irfanview-based raw conversion, with the exception of **-transCmdList** that holds Imagemagick transformation commands as TCL list, like {-gamma 1.2}.

Definition of the similar command but without image-orientation support differs only in two arguments: name of command building procedure is "::iconv::build_cmd__default_convert_img", and **-previewPath** argument should not appear at all.

Postprocessing procedures for LazyConv.

By LazyConv's convention, postprocessing procedure gets two arguments: input file path and

output specification (see “Output specifications”), performs the conversion in whatever way, returns 1 on success, 0 on error.

Standard postprocessing procedures are located in TCL/Work/imageproc/postconvert.tcl; you may use them as templates when defining yours. Each standard procedure provides a color-conversion string that follows the standard of Imagemagick command-line tools (see Imagemagick web help for more info).

Example: “:postconvert::contrP1dark1” procedure mentioned in “Irfanview based raw conversion commands”:

```
proc ::postconvert::contrP1dark1 {inpPath outspecArrName} {
        upvar $outspecArrName outSpec
        return  [im_convert $inpPath outSpec {-sigmoidal-contrast 3,60%}]
}
```

When defining your own postprocessing procedures, you probably want to change only the **bold** parts in the above example.

Changing the default conversion command.

Consider doing it, if you find yourself applying the same color correction to the majority of images.

To change the default conversion command for one session, edit its inp_config.tcl file after you created it as a part of work-area preparation.

Change from:

set CONVERT_CMD_NICKNAME **CONVERT_ROTATE_MRW_DEFAULT**

to

set CONVERT_CMD_NICKNAME **<your preferred command nickname>**

To change the default conversion command continuously, edit LazyConv's source-code files that contain wrappers for conversion engines:

- TCL/Work/tool_wrap_main/wrap_rconv_iview.tcl (Irfanview wrapper)

  Change from:

  ```
  proc ::irfanview::get_default_convert_cmd {} {
     return  "CONVERT_ROTATE_MRW_DEFAULT"
  }
  ```

  to

  ```
  proc ::irfanview::get_default_convert_cmd {} {
     return  "<your preferred command nickname>"
  }
  ```

- TCL/Work/tool_wrap_main/wrap_rconv_dcraw.tcl (dcraw wrapper)

  Change from:

  ```
  proc ::dcraw::get_default_convert_cmd {} {
     return  "CONVERT_ROTATE_MRW_DEFAULT"
  }
  ```

  to

  ```
  proc ::dcraw::get_default_convert_cmd {} {
     return  "<your preferred command nickname>"
  }
  ```

  Of course, the command referenced in any of the two ways should exist in the system.

Defining image-viewing commands.

Actually I wanted to drop this chapter in the first version; but I will provide basic facts.

Viewing commands in LazyConv have some similarity to the conversion commands – they build OS- and viewer – specific command line to be run on a specified image.

Here is the definition as appears in TCL/Work/tool_wrap_main/wrap_view_iview.tcl:

```
xx::define_cmd SHOW_PREVIEW_IMAGE \
   [list \
      -Description "shows one image" \
      -BuildProc "::irfanview::build_cmd__show_one_image" \
      -ONameSuffix "_DUMMY1" -inpFile "@iName@" \
      -outspecArrName "@outSpec@" -iniDir "$INIDIR_DEFAULT"]
```

Of course, output name suffix and output specification are dummy. The TCL procedure used for command definition is the same as described in “Defining conversion commands”.

Usage scenarios' demonstration.

This chapter describes simple and advanced usage scenarios that not only are supported by LazyConv, but their demo-s are part of its testing infrastructure.

The LazyConv distribution includes TCL file TCL/Work/cnv_test/demo_tests.tcl that contains several procedures that simulate LazyConv conversion sessions. Each such procedure includes both the commands a user would normally type, and additional verification code that verifies correctness of their execution. The latter have increased indent, so that the reader can easily concentrate on the former.

Following are kinds of commands found in each demo procedure:

1. Creation of working directory and bringing the input files.
    - the user would do it manually, so the procedure used in the demo isn't interesting
2. Work-area preparation.
    - performed by **run_setup_work_area** just as the user would do
    - ignore the tricks used to obtain inputs' origin and directory names
3. Converter initialization.
    - by sourcing proper LazyConv initialization script (setup_rconv.tcl, setup_rconv_dcraw.tcl, or setup_iconv.tcl)
    - ignore the trick used to obtain the right initialization script for the inputs' origin
4. Setting the work area to the one created earlier.
    - by **set_work_area**, as usual
5. Creation of previews for all the inputs.
    - by **run_preview_all**, as usual
6. One or more iterations of copying/moving/rotating/deleting the preview images and making corrected previews.
    - models what user normally does when sorting the previews
    - TCL standard commands used for manipulating image files
    - color-corrected previews produced by **run_correction**, as usual
7. Final conversion.
    - by **run_conversion**, as usual
8. Detection of unused input files
    - by **run_process_unlisted**, as usual
    - these are candidates for deletion

Sessions' descriptions in this chapter contain commands and comments; the latter typed in *italic* and preceded by #.

Ordinary session.

Procedure ***::demo_tests::just_run_it*** in TCL/Work/cnv_test/demo_tests.tcl .

The most straightforward session possible.

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

set_work_area <work_dir>

run_preview_all

 # imitate image sorting for corrections

 # here preview files are MOVED into correction dirs

file rename -force -- <preview_image_1> <correction_dir_1>

file rename -force -- <preview_image_2> <dir_for_manual_correction>

run_correction

 # remove 2 images; both stay in same place as created

file delete -- <image_to_delete_1>; file delete -- <image_to_delete_2>

run_conversion; *# converts remaining images*

run_process_unlisted; *# renames unused input (raw) files*

Dealing with duplicated previews.

Procedure ***::demo_tests::duplicate_previews*** in TCL/Work/cnv_test/demo_tests.tcl .

Imitates partial preview+correct session where the following valid scenarios occur:

- previews are copied into correction dirs (instead of being moved)
- the same image is scheduled for two different corrections
- the same correction of the same image performed twice

The point is that when you schedule an image for correction, you may either move its preview into the correction directory, or copy it there.

- you have to copy when ordering several different corrections at the same time
- at the end of **run_correction**, all moved/copied previews are returned to the default output directory; if you duplicated a preview, LazyConv will encounter a conflict and spend some time on validating that the previews with the same names have the same contents
    - this happens when you either copy a preview to a correction directory, or order the same correction twice (in different calls of **run_correction**)
    - e.g. one preview stayed in the default output directory, another one either brought from a correction directory or just created
    - if files are equal, one of them is deleted; otherwise error is reported
    - so, if you need a single correction, move the preview, do not copy

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

    -inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

set_work_area <work_dir>

run_preview_all

 # imitate image sorting for corrections

 # here preview files are COPIED into correction dirs

file copy -force -- <preview_image_1> <correction_dir_1>

file copy -force -- <preview_image_2> <dir_for_manual_correction>

run_correction

 # imitate ordering same image for two corrections:

file copy -force -- <preview_image_1> <correction_dir_1>

file copy -force -- <preview_image_1> <correction_dir_2>

run_correction; *# should process 1 image and skip 1 existing image*

Using two groups of image-sorting directories.

Procedure ***::demo_tests::run_with_2_dirs*** in TCL/Work/cnv_test/demo_tests.tcl .

Demonstrates LazyConv's handling of a bunch of images being split into subsets.

Reasons for the user for doing this:

- to work with smaller sets of images
- to sort images by topics (landscapes, people, architecture, locations)
- to sort images by quality – ranging from “takers” down to “throw-outs”

The essence of the feature in LazyConv:

1. **run_correction** will write each preview into root directory of its subset
2. **run_browse_previews_in_dir** naturally looks at one directory and thus sees a single subset

The feature is enabled by:

1. Creating work-area with several sorting-directory groups.

   run_setup_work_area ... **-sort_dir_names {**<Name1> ... <NameN>**}**

2. Moving previews from the default output directory (specified through “**-work_dirs_root**”) into the root directories of these several sorting groups.

The structure of the working directory with two subsets “Set1” and “Set2” is shown below.

```
CONV2DIR> dir out
        out:
        Set1  Set2
CONV2DIR> dir out/Set1 out/Set2
out/Set1:
        DoBrightP1      DoBrightP2      DoContrMinus1   DoContrMinus2   DoContrPlus1
```

DoContrPlus1Dark1 DoContrPlus2 DoContrPlus2Dark1 DoDark DoDesatBright

KeepAsIs Manual

out/Set2:

DoBrightP1 DoBrightP2 DoContrMinus1 DoContrMinus2 DoContrPlus1

DoContrPlus1Dark1 DoContrPlus2 DoContrPlus2Dark1 DoDark DoDesatBright

KeepAsIs Manual

When **run_preview_all** makes the previews, they go to the default output directory, then you should move all of them to the sub-groups' roots. The correction directories for a group are located under the root directory of this group. Upon creating the corrected preview, **run_correction** puts it and the original preview into the group's root directory.

Important: do not **run_preview_all** any more for this session; it would create second set of the same images.

run_conversion, on the contrary, puts its outputs into a single Final/ directory under the default output directory.

Both **run_correction** and **run_conversion** try to process images in all the sets. To make a set be ignored, just leave all its previews in its root directory - **run_correction** will have nothing to do with it then. For **run_conversion** – do not run it until finalizing corrections of all the images in all the sets. And **run_browse_previews_in_dir** anyway gets the directory as an argument.

Here is the scenario of ::demo_tests::run_with_2_dirs demo.

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir> \

-sort_dir_names {Group1 Group2**}**

set_work_area <work_dir>

run_preview_all

 # distribute preview images between sorting directories:

file rename -force -- <preview_image_1> <root_of_Group1>

file rename -force -- <preview_image_2> <root_of_Group1>

file rename -force -- <preview_image_3> <root_of_Group2>

file rename -force -- <preview_image_4> <root_of_Group2>

 # imitate image sorting for corrections, note one image scheduled for two corrections

file copy -force -- <preview_image_1> <root_of_Group1/correction_dir_1>

file rename -force -- <preview_image_1> <root_of_Group1/dir_for_manual_correction>

file rename -force -- <preview_image_2> <root_of_Group1/correction_dir_2>

file rename -force -- <preview_image_3> <root_of_Group2/correction_dir_2>

file rename -force -- <preview_image_4> <root_of_Group2/dir_for_keep_as_is>

run_correction

 # remove all occurrences of <preview_image_1> except those in manual dir

file delete -- <root_of_Group1>/<preview_image_1>

file delete -- <root_of_Group1>/<corrected_preview_image_1>

run_conversion; *# converts remaining images: 1 under Group1/, 4 under Group2/*

run_process_unlisted; *# there are no unused input (raw) files to rename this time*

Using external preview files.

Procedure ***::demo_tests::external_previews*** in TCL/Work/cnv_test/demo_tests.tcl .

Imitates session that uses external preview files. (I.e. the preview files don't have LazyConv's command-bound suffixes in their names.) Such preview files should result in default conversion.

This feature is useful when you performed initial image sorting for a bunch of images before starting a LazyConv session for it.

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

set_work_area <work_dir>

run_preview_all

 # rename all the previews – discard LazyConv – specific suffixes

 # ... (renaming commands not shown) ...

run_process_unlisted; *# expected to make no impact*

 # imitate image sorting for corrections (copy one and move another)

file copy -force -- <preview_image_1> <correction_dir_1>

file rename -force -- <preview_image_2> <dir_for_manual_correction>

run_correction

 # delete two previews

file delete -- <preview_image_3>; file delete -- <preview_image_4>

run_conversion; *# should process 2 images*

run_process_unlisted; *# should process 2 images*

Copying input files through their previews.

Procedure ***::demo_tests::copy_inputs_by_previews*** in TCL/Work/cnv_test/demo_tests.tcl .

As a by-product, LazyConv can help you selecting a subset of input (raw) files given their previews. Let's say you have 200 input files and previews of 30 of them; you want to select inputs for these 30 previews. In LazyConv you do it by placing your 30 previews into manual-correction directory and invoking **run_correction**. You'll get the copies of corresponding input files in the same manual-correction directory.

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

set_work_area <work_dir>

run_preview_all

 # delete 2 preview images and order 2 others for copy input

file delete -- <preview_image_1>; file delete -- <preview_image_2>

file rename -force -- <preview_image_3> <dir_for_manual_correction>

file rename -force -- <preview_image_4> <dir_for_manual_correction>

run_correction; *# should copy 2 inputs into <dir_for_manual_correction>*

 # now you have inputs for <preview_image_3> and <preview_image_4> in <dir_for_manual_correction>

run_process_unlisted; *# should process 2 images*

Keeping inputs in nested directories.

Procedure ***::demo_tests::nested_inpdirs_keep_as_is*** in TCL/Work/cnv_test/demo_tests.tcl

Demonstrates the following features:

- ability to keep input (raw) files in sub-directories under one root

  for example, you are processing a bunch where all inputs are grouped by taking date; just put all of them under one root directory

- keep-as-is sorting directory - for the previews that are just right – preview images there are ignored by **run_correction** and you don't see them while browsing previews in the default output directory

  source <appropriate_LazyConv_setup_script>

  run_setup_work_area -origin <origin> \

  -inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

  set_work_area <work_dir>

run_preview_all; *# nothing interesting yet – inputs are still in <input_dir>*

 # delete all the previews – we'll remake them from inputs in subdirectories

 # ... (deletion commands not shown) ...

 # move input files into subdirectories of <input_dir>

file mkdir <input_dir>/Dir1 <input_dir>/Dir2

file rename -force -- <input_image_1> <input_dir>/Dir1

file rename -force -- <input_image_2> <input_dir>/Dir1

file rename -force -- <input_image_3> <input_dir>/Dir2

file rename -force -- <input_image_4> <input_dir>/Dir2

run_preview_all; *# makes previews while inputs are somewhere under <input_dir>*

 # imitate image sorting for corrections; one of them goes to keep-as-is directory

file rename -force -- <preview_image_1> <correction_dir_1>

file rename -force -- <preview_image_2> <keep-as-is_dir>

run_correction; *# should process 1 image*

 # delete 2 preview images: <preview_image_1> (default) and <preview_image_4>; both stay in same place as created

file delete -- <preview_image_1>; file delete -- <preview_image_4>

run_conversion; *# should convert 3 images: <corrected_preview_image_1>, <keep-as-is_dir>/<preview_image_2> (hidden), <preview_image_3>*

run_process_unlisted; *# should process 1 image (input of <preview_image_4>)*

Restoring default previews.

Procedure ***::demo_tests::restore_default_previews*** in TCL/Work/cnv_test/demo_tests.tcl

Demonstrates the ability to re-create default image preview (like the one created by **run_preview_all**)

for example, you may have deleted the default preview, but then decided you need it

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

set_work_area <work_dir>

run_preview_all

imitate image sorting for corrections: move all 4 previews into one correction

directory

file rename -force -- <default_preview_image_1> <correction_dir_1>

file rename -force -- <default_preview_image_2> <correction_dir_1>

file rename -force -- <default_preview_image_3> <correction_dir_1>

file rename -force -- <default_preview_image_4> <correction_dir_1>

run_correction ; # creates 4 non-default (color-corrected) previews

remove all default previews: all stay in same place as created
file delete -- <default_preview_image_1>

file delete -- <default_preview_image_2>

file delete -- <default_preview_image_3>

file delete -- <default_preview_image_4>

copy all color-corrected previews to trigger recreation of the default ones
file copy -force -- <corrected_preview_image_1> <***default_preview_di***r>

file copy -force -- <corrected_preview_image_2> <***default_preview_dir***>

file copy -force -- <corrected_preview_image_3> <***default_preview_dir***>

file copy -force -- <corrected_preview_image_4> <***default_preview_dir***>

run_correction ; # creates 4 default previews

run_process_unlisted; *# though there are no unused input (raw) files to rename*

Basic white-balance overrides.

Procedure ***::demo_tests::use_override_wb*** in TCL/Work/cnv_test/demo_tests.tcl

Demonstrates using one white-balance preset for a work-area. For more user-friendly approach look at “Mainstream usage of white-balance overrides”.

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

    -inp_dirs_root <input_dir> **-work_dirs_root** <work_dir>

set_work_area <work_dir>

check whether the underlying converter supports WB overrides; return otherwise

activate white-balance preset standardly shipped with LazyConv

::camera_data::define_camera__*SAMPLE_MINOLTA_A2*

set wbP [::camera_data::get_camera_wb_preset *SAMPLE_MINOLTA_A2 Cloudy*]

::dcraw::set_wb_override $wbP

run_preview_all; *# will create previews with the overridden white balance*

 # imitate image sorting for corrections: MOVE one preview into a correction directory, another one – into manual-processing directory:

file rename -force -- <preview_image_1> <correction_dir_1>

file rename -force -- <preview_image_2> <dir_for_manual_correction>

run_correction*;* *# should process 1 image with WB override*

run_conversion*;* *# should process 4 images with WB override*

run_process_unlisted; *# should have no effect*

::dcraw::cancel_wb_override; *# deactivate white-balance override*

remove all previews and final images with overrides

run_preview_all; *# will create previews without WB override*

just verify that previews without WB overrides are created correctly

remove all previews without overrides

 # create WB override file in work-area root directory:

camera_data::write_wb_ovrd_file_in_dir SAMPLE_MINOLTA_A2 Cloudy <work_area_root_directory>

 # activate another WB override at lower priority - to verify that it's ignored:

set wbD [::camera_data::get_camera_wb_preset SAMPLE_MINOLTA_A2 Flash]

::dcraw::set_wb_override $wbD

puts "\n==== Run make-preview for $inpImgNum images with WB override through a file in '$configDir'"

 # create previews with WB override through a file in the work area root directory

run_preview_all

 # order all the previews for the same correction:

file rename -force -- <preview_image_i> <correction_dir_1>; # i = 1...4

run_correction*;* *# should process 4 images with WB override through a file in the work area root directory*

 # convert 8 images with WB override through a file in the work area root directory:

run_conversion

Using sorting directories with white-balance overrides.

Procedure ***::demo_tests::camera_wb_ovrd_dirs*** in TCL/Work/cnv_test/demo_tests.tcl

Demonstrates custom scenarios of using multiple white-balance presets in LazyConv. For more user-friendly approach look at “Mainstream usage of white-balance overrides”.

source <appropriate_LazyConv_setup_script>

 # create work area with sort-root directories for registered WB presets of one camera

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir> \

-wb_ovrd_dirs_for_camera <registered_camera_name>

set_work_area <work_dir>

check whether the underlying converter supports WB overrides; return otherwise

run_preview_all; *# create previews with the default white balance*

 # move 2 images into AutoWB/ sort-root directory

file rename -force -- <preview_image_2> <root_dir_for_autoWB>

file rename -force -- <preview_image_3> <root_dir_for_autoWB>

delete another 2 images, so that they don't interfere

file delete -force <preview_image_0>

file delete -force <preview_image_1>

 # imitate image sorting for corrections under AutoWB/ preview-root directory:

 # - put one preview into both a correction directory, and manual-processing directory

 # - move another preview into a different correction directory

file copy -force <AutoWB/preview_image_2> <AutoWB/correction_dir_1>

file rename -force <AutoWB/preview_image_2> <AutoWB/dir_for_manual_correction>

file rename -force <AutoWB/preview_image_3> <AutoWB/correction_dir_2>

run_correction*; # should process 2 images and order 1 for manual processing; NO WB override expected*

 # re-create all the default (no WB ovrd) previews - remove all, then run_preview_all

file delete -force <post_correction_preview_image_i>; # i = 0...4

run_preview_all; *# will create previews with the default white balance*

 # move 2 previews into sort-root directory with an active override (Cloudy/)

file rename -force -- <preview_image_0> <root_dir_for_Cloudy>

file rename -force -- <preview_image_1> <root_dir_for_Cloudy>

delete another 2 images, so that they don't interfere

file delete -force <preview_image_2>

file delete -force <preview_image_3>

 # order both remaining preview for default processing with their chosen white balance

 # MOVE these 2 previews into <Cloudy/default_preview_dir>

file rename -force -- <Cloudy/preview_image_0> <Cloudy/default_preview_dir>

file rename -force -- <Cloudy/preview_image_1> <Cloudy/default_preview_dir>

run_correction; *# should create 2 default previews with WB override*

 # delete the 2 previews without overrides, so that they don't interfere

 # the new images now have the names of <preview_image_0> and <preview_image_1>; delete all images with different names

file delete -force <renamed__ original_preview_image_0>

file delete -force <renamed__ original_preview_image_1>

 # imitate image sorting for corrections; everything with "Cloudy" white balance

 # order <preview_image_0> for 2 different corrections, hide <preview_image_1> from corrections

file copy -force -- <Cloudy/preview_image_0> <Cloudy/correction_dir_1>

file copy -force -- <Cloudy/preview_image_0> <Cloudy/correction_dir_2>

file rename -force -- <Cloudy/preview_image_1> <Cloudy/keep-as-is_dir>

run_correction; *# should process 2 images and keep 1 as is*

 # hide all existing previews from correction

file rename -force <Cloudy/preview_image_i> <Cloudy/keep-as-is_dir>; # i = 0...3

run_conversion; *# should process 4 images with WB override*

run_process_unlisted; *# should rename 2 input (RAW) files that have no previews*

Mainstream usage of white-balance overrides.

Procedure ***::demo_tests::use_remake_wb_ovrd_previews*** in TCL/Work/cnv_test/demo_tests.tcl

Demonstrates using white-balance presets in LazyConv by means of **run_remake_wb_ovrd_previews** command.

source <appropriate_LazyConv_setup_script>

run_setup_work_area -origin <origin> \

-inp_dirs_root <input_dir> **-work_dirs_root** <work_dir> \

-wb_ovrd_dirs_for_camera <registered_camera_name>

set_work_area <work_dir>

check whether the underlying converter supports WB overrides; return otherwise

run_preview_all; *# create previews with the default white balance*

 # move 2 images into Cloudy/ sort-root directory (for "Cloudy" white balance)

file rename -force -- <preview_image_0> <root_dir_for_Cloudy>

file rename -force -- <preview_image_1> <root_dir_for_Cloudy>

 # move 2 images into AutoWB/ sort-root directory (no WB override there)

file rename -force -- <preview_image_2> <root_dir_for_autoWB>

file rename -force -- <preview_image_3> <root_dir_for_autoWB>

run_remake_wb_ovrd_previews; *# should remake 2 images for "Cloudy" WB*

 # delete one of the images with "Cloudy" WB to save time

file delete -force <root_dir_for_Cloudy /preview_image_0>

 # imitate image sorting for corrections:

 # - 1 image for correction with WB override; 2 images for correction without WB override

file rename -force -- <Cloudy/preview_image_1> <Cloudy/correction_dir_2>

file rename -force -- <AutoWB/preview_image_2> <AutoWB/correction_dir_1>

file rename -force -- <AutoWB/preview_image_3> <AutoWB/correction_dir_2>

run_correction; *# should create 3 new images; 1 of them with WB override*

 # delete all previews with WB overrides

file delete -force <Cloudy/preview_image_*>

 # test remake of CORRECTED image in a single directory with WB-override

 # - delete one default image in AutoWB/ and move its correction into Cloudy/

file delete -force <AutoWB/<preview_image_3>

file rename -force <AutoWB/corrected__preview_image_3> <root_dir_for_Cloudy>

run_remake_wb_ovrd_previews Cloudy/; # should remake 1 image under Cloudy/

delete the single existing image with WB override

file delete -force <Cloudy/corrected__preview_image_3>

 # we are left with 2 versions of one input image – both without WB overrides

run_conversion; *# should make final conversions of 2 images without WB overrides*

run_process_unlisted; *# should rename 3 input (RAW) files that have no previews*

Running the demo tests.

Read this chapter if you want to run the session-scenarios described above through your installation of LazyConv.

It's assumed that you arranged all test-related settings as described in "Installation steps".

The same group of demo-tests could be run on one of the following kinds of input files:

MINOLTA, TIFF and JPEG. It's recommended to open a new TCL console for each run.

1. Start new TCL console as described in “Starting TCL interpreter”. (All following commands are typed in it.)
2. Create a working directory for the testing

   file mkdir c:/TEST_LZCONV/
3. Load LazyConv's code (path in the example matches my installation):

   source C:/ANY/LazyConv/TCL/Work/setup.tcl; package require cnv_test
4. Configure desired raw conversion engine:

   for Irfanview:

   ::cnv_test::set_tst_raw_cnv_setup_file setup_rconv.tcl

   ::cnv_test::set_tst_raw_cmd_defs_file wrap_rconv_iview.tcl

   for dcraw:

   ::cnv_test::set_tst_raw_cnv_setup_file setup_rconv_dcraw.tcl

   ::cnv_test::set_tst_raw_cmd_defs_file wrap_rconv_dcraw.tcl
5. Run the tests for JPEG input (fastest):

   ::cnv_test::set_tst_origin JPEG; **::demo_tests::run_all** "$OK_TCLSRC_ROOT" {C:/TEST_LZCONV/} {DTest}
6. In order to run tests on MINOLTA or TIFF files, substitute the origin in the command above.

If the tests run smoothly, you should receive the following test on your console:

```
%%% Test <just_run_it>   passed
%%% Test <duplicate_previews>          passed
%%% Test <run_with_2_dirs>     passed
%%% Test <external_previews> passed
%%% Test <copy_inputs_by_previews>          passed
%%% Test <nested_inpdirs_keep_as_is>       passed
```

Leftovers.

Performance data.

Here are some performance measurements for long-running LazyConv commands. Input images came from Sony Alpha100 10-megapixel DSLR. The computer has Intel Core Duo 6600 2.4 GHz processor and 2Gb of 800 MHz RAM; OS is Windows Vista. All the images were rotated to achieve the worst-case scenario.

Command	Number of inputs to browse	Number of conversions or renames made	Elapsed time (min)	Images per a minute
run_preview_all	428	428	45	9.5
run_process_unlisted	428	0	0	0
run_correction (no WB override)	428	428	73	5.86
run_remake_wb_ovrd_previews	428	428	82	5.22
run_correction (with WB override)	428	428	72	5.94
run_conversion	428	428	102	4.2

Future directions.

I plan the following enhancements in LazyConv; of course depending on how it's met by the community:

1. Incorporate EXIF metadata handling if appropriate engine becomes available. This will allow:
    - Automatic handling of image orientation.
2. More camera-raw formats
3. Enable image resizing through output specification.
4. Define more conversion/correction commands.
5. More features may come if requested and doable.

Disclaimer: I wrote LazyConv in after-hours, so my development bandwidth is pretty limited. Well, unless I get fired from the main job or start being paid for LazyConv :).