

ImageMagick v6 Examples -- Usage under Windows

Index

[Introduction](#)

- [What's the use of an IM script on my Windows PC?](#)
- [Possible environments for IM's command line tools](#)
- [Running scripts effectively](#)
- [The Convert issue](#)
- [Character Encoding](#)
- [Installing ImageMagick under Windows](#)
- [Specialities and pitfalls](#)
- [Getting help](#)
- [Auxiliary programs and alternatives](#)

[Using Cygwin](#)

[Using the DOS Shell and Batch Files](#)

- [Converting Scripts: UNIX Shell to Window DOS](#)
- [Filename Handling in Batch files](#)
- [FOR Loops](#) -- Batch Processing Several Files
- [Batch processing a \(sub-\)directory tree](#)
- [Batch Processing an Arbitrary Number of Files](#)
- [Re-using the Output of an IM command](#)
- [Processing Single Line Output](#)
- [Performing Calculations](#)
 - [Using IM's FX Expressions](#)
 - [Using the SET command](#)
 - [Using Other External Calculators](#)
- [Editing, Debugging and Runtime Error Testing](#)
- [Optimising execution time](#)
- [Guidelines for Batch Programming](#)
- [Summing it up](#)

[Visual Basic Script \(VBS\)](#)

- [A Basic Example: Lens Correction](#)
- [Working with Several Files](#)
- [Working with Text Files](#)
- [Piping](#)
- [Testing and Debugging VBScripts](#)

[Further Information](#)

Most of the commands in IM Examples were written specifically with LINUX and UNIX shell scripting in mind, as these systems are designed with batch processing and network servers in mind. However, more and more users of ImageMagick want to use it from the Windows Environment. This section provides details and examples of how you can use IM in that environment and, more important, how to convert a UNIX shell command (as used in the rest of the IM Examples) into its Windows DOS equivalent.

I wish to express specific thanks to Wolfgang Hugemann <ImageMagick_AT_Hugemann.de> who completely re-wrote the original notes and expanded them to cover a much larger range of topics, of specific concern to window users. What you see here is his work, and IM users are indebted to him for his time and patience.

Introduction

What's the use of an IM script on my Windows PC?

The following examples basically assume that you run IM on a Windows desktop computer, probably attached to a network. Well, there are a lot of readily available image manipulation programs, such as Adobe Photoshop, Corel's Paint Shop Pro, IrfanView (<http://www.irfanview.com/>) and even GIMP (<http://www.gimp.org/>). So why should you bother to perform image processing by IM's command line programs and scripts?

The genuine advantage of using ImageMagick instead of a mouse-driven interface is that you can completely automate routine manipulations, either for single files or for bulks of files. Tasks such as:

Bulk format conversion

This is offered by quite a lot of Windows programs, such as IrfanView. However, IM's versatility when it comes to image formats is unsurpassed. You can for instance convert all the pages of a PDF into a series of JPEGs (if GhostScript is installed on your computer).

Shrinking and preprocessing digital photographs

When embedding digital photographs into a word processor document, you should usually reduce their resolution, such that the document can be printed faster. The same holds if you convert the document into a PDF via a PDF printer driver such as [FreePDF](#). Preprocessing could also comprise colour and lens correction routines.

Placing your logo into a bulk of digital photographs

Applying a series of operations to a bulk of digital photographs

Having worked out a series of working steps by use of a mouse-driven program, you might wish to automate these steps for future bulk processing. However, script languages (such as Adobe's Action Script) are not very common in Windows image processing programs.

Combining several images to a catalog image

Although some of these tasks (especially bulk-shrinking) are also offered by other freeware programs (especially by IrfanView's batch processing), you are never free of choice in what processing steps to apply: you have to live with those offered by the program. For instance, IrfanView's batch processing offers to place a text string into a bulk of photographs, but not a logo. It also offers to change the gamma value, but the histogram of the photograph cannot be clipped at its ends (as done by "[-contrast-stretch](#)" in IM, see [Normalize and Contrast Stretch](#)).

IM scripts are especially suited for productive use in a company network, because ready-made scripts can be applied by anyone – end users do not necessarily have to know about what's going on in the background. Thus standard workflow steps on images can be completely automatised (and really standardised). Several of the scripts presented in the following were derived for productive use in our small company (working in the field of accident reconstruction).

I am neither an outstanding Windows script programmer nor most familiar with IM's command line tools. There are probably more elegant approaches to some of the problems treated in the following. The points I want to make are:

- to demonstrate some basic techniques of Windows script programming with IM's command line tools.
- to prove that the use of IM-based scripts is neither art for art's sake nor an academic pastime.
- to show that IM's command line tools can do real work in a local network (and not only on webserver).

As in the rest of IM's Example pages, we will only use IM's command line tools and leave its various programming interfaces aside. The scripts are intended to run within a local network with drive letters assigned to each network drive. Most of the scripts are intended to be run on the client computers of that network, few aim at the network (file) server.

Possible environments for IM's command line tools

Under Windows, simple IM commands are usually run in the Windows Command Shell (a "DOS Shell" run by starting `cmd.exe`). For complex operations, performed in a lengthy command line or in a series of command lines, you will better write a script. For a series of simple commands, this will most probably be a DOS batch file, executed in the Windows Command Shell. This approach, however, has its shortcomings, as the batch file command set is rather limited in comparison to those of common UNIX command shells.

When running IM under Windows, you basically have the following alternatives:

The Windows command shell ("DOS window")

This is run by `cmd.exe` (32-bit mode) on Windows NT 4.0, Windows XP and later versions and is present on any Windows computer. See [Using the DOS Shell and Batch Files](#), as well as the special note about the [The Convert Issue](#).

Cygwin

A bash-like command shell (<http://www.cygwin.com/>). When using this shell, the IM examples presented in the rest of this usage section can be run exactly as given, as you have access to a UNIX style command line shell. See [Using Cygwin](#) below.

The Windows Script Host

The Windows Script Host is based on the .COM technology. It is present on any contemporary Windows computer and WSH scripts are much more powerful than simple DOS batch files. The Windows Script Host offers several programming interfaces, with VBScript (Visual Basic Script) and JScript (Java Script) being the most common. The IM command line tools can be invoked by using the DOS shell commands Run or Exec of the Shell object. See [Visual Basic Script \(VBS\)](#) below.

The Windows Powershell

The much more powerful successor of the ancient DOS shell, based on the .NET 2.0 technology. The Powershell shipped with Windows 7 and is run by `powershell.exe`. It can be downloaded for Windows XP and Vista at Microsoft's website.

Running scripts effectively

Let's assume you have a perfect Windows script (a DOS batch file, a VBScript, or whatever) that takes the name(s) of the input file(s) as command line parameter(s), performs some manipulation and spits out the result as a single image or a series of images. You surely won't like to start a DOS command shell (or an alternative) everytime and provide the script with the filenames by typing them. To avoid such cumbersome ways of proceeding, you can basically use **Drag & Drop** or **SendTo**:

When using **Drag & Drop**, you place the DOS batch file or the VBScript (or whatever) in a location that is easily accessible, like the desktop or the directory which holds the image files to be processed. You then select the files to be processed in the Windows Explorer and just drop them onto the script file. The filenames will be handed over to the script as the command line parameters and can be referred to in the script.

As an alternative, you can place your script (or a link to it) in the **SendTo** folder. The programs in this folder appear in the context menu of the Windows Explorer when right-clicking in the Explorer's file pane. Again, the names of the selected files are handed over to the script as command line parameters. The SendTo folder is named `SendTo`. Its location seems to move with each new Windows version. A bullet-proof way to find it is typing `shell:sendto` into the run box.

A single command line under Windows XP or later can be 8192 (= 2¹³) characters long. So if you invoke an IM tool directly from the command line, either directly or via a batch file that names all the files needed directly, you will hardly run into this limitation. Drag & Drop however uses the ExecShell routine, which is limited to strings of "only" 2048 (= 2¹¹) characters. As all files are passed with fully qualified filenames (i.e. drive + path + name), this can become a problem for batch files and VBScripts run via WSH when the path names become too long. This errors cannot be properly handled by the script once it occurs, as the error occurs **before** the script is actually run. The solution under Windows XP is usually to place the files in a location where the pathnames are shorter.

The Convert issue

IM's Windows installation routine adds IM's program directory to the search path, such that you can call IM's command line tools directly from the command prompt, without providing a path name. However, the names of IM's command line tools are rather unspecific (e.g. Convert, Identify, Compare ...) which provokes name conflicts with other programs.

Especially, Convert is a Windows system tool, located in the Windows system directory (`c:\Windows\system32\convert.exe`), which converts the FAT32 file system into the now common NTFS. But there are also other programs named "convert.exe", for example the Delphi report converter utility.

The FAT → NTFS convert tool was first shipped with Windows XP and generated a name conflict with IM's command line tool "convert" as IM's program directory was **appended** to the DOS search path (i.e. the PATH environment variable), the system tool was found first and simple calls to "convert" in older scripts weren't resolved correctly. Current versions of IM's Windows setup program however place IM's program directory at the head of the search path, thereby insuring that in case of conflicting names, IM's command line tool is usually found first. However, other utilities with the same name (e.g. Delphi's report converter) ran into the same problem and went for the same solution, i.e. placing their program path at the head of the path variable, which means that this solution is not bullet-proof: If Delphi is installed after IM, a simple call of Convert will invoke the report converter tool, not IM's Convert.

The introduction of the Convert tool with Windows XP caused a lot of legacy scripts to crash. The common solution was to rename IM's Convert tool to something else, such as "IMconvert" (Note that you can not rename the system command, as the next Windows service pack would probably just restore it, ignoring the renamed version.) This solution, although unnecessary now, can still be found all over the Internet.

The best solution to avoid possible future name conflicts is to call IM's command line tools by their full pathname in any script. That is, storing its location in a variable or a constant. So every batch file should start with lines like

```
SETLOCAL EnableDelayedExpansion
SET IMCONV="%PROGRAMFILES%\ImageMagick\Convert"
...
%IMCONV% -size 128x128 xc:white test.gif
```

The code assumes that IM was installed in a folder named "ImageMagick" below the program folder, which is **not** the standard naming of its installation folder. (See [Installing ImageMagick under Windows](#) for details.) %PROGRAMFILES% is an environment variable which expands to the name of the program directory, i.e. "Program Files" in the English Windows version and "Programme" in the German Windows version. SETLOCAL will limit the definition of new environment variables (such as IMCONV) to the scope of the batch file. EnableDelayedExpansion is not really needed over here, but it is a good habit to use this option each time you are using SETLOCAL; see [Guidelines for Batch Programming](#), for details. There are more sophisticated and bullet-proof ways to find out about IM's installation folder, which will be treated in [Editing, Debugging and Runtime Error Testing](#). The equivalent VB-Script code would be something like

```
Set wsh = WScript.CreateObject("WScript.Shell")
IMconv = wsh.ExpandEnvironmentStrings("%PROGRAMFILES%") & "\ImageMagick\Convert"
```

For reasons of simplicity, we will not use this code everytime in the following, but you should keep this in mind as a good habit. For a good alternative summary and solutions to this problem see [Ron Savage: MS Windows and convert.exe](#).

Character Encoding

ImageMagick encodes strings in Unicode, more precisely in [UTF-8](#). To the contrary, DOS uses codepages to encode characters (mostly in 8-bits). This generates problems when writing strings into images, such as when working with 'label' or '-title': When using non-ASCII characters, things will go wrong in the easy approach.

For example trying to create a label of German umlauts such as 'ä', 'ö', 'ü', you can simply use the following in Linux...

```
convert label:äöü test.png
```

But this would not create the desired string under windows. You can however read and UTF-8 coded string from a textfile:

```
convert label:umlauts.txt test.png
```

and you can even create this textfile by the use of "echo, if you switch the codepage to UTF-8 in advance:

```
CHCP 65001
ECHO äöü > umlauts.txt
convert label:umlauts.txt test.png
```

But if you want to process the output of a DOS command, for example when trying to title an index print of the JPEGs contained in a certain directory with the name of this directory, you are in trouble. Switching to codepage 65001 will not work with most of the DOS commands, especially when looping through directory trees. And switching the codepage to and fro between, say 1252 (West European Latin) and 65001 will usually not work either or become at least rather tricky.

The safest approach is to convert strings when they are needed, using an external command line program, such as "Iconv.exe", a UNIX tool which is also available for Windows. Download the setup file from [SourceForge](#) and install the files into the standard directory C:\Program Files\GnuWin32. Then dump the output of the DOS command to a text file in your script and convert that file to UTF-8 in the following:

```
CHCP 1252
DIR /B äöü.jpg > temp.txt
"C:\Program Files\Gnuwin32\bin\iconv.exe" -f ISO-8859-1 -t UTF-8 temp.txt > title.txt
convert label:@title.txt äöü.jpg -append äöü_labelled.jpg
```

The parameters tell Iconv to transcode from ISO-8859-1 (codepage 1252) to UTF-8. Iconv writes its output to stdout, so that you have to redirect it to a file in order to use it with 'label'. Please note that dumping the output to a textfile is recommendable anyway, because it tells ImageMagick to interpretate the files contents literally, without taking the Windows backslash ("\") for an escape character.

Of course, the code in the above example does not make much sense as it is presented here for demonstration purposes. In a practical DOS batch file, the file name will probably be generated in a FOR loop. A practical example is given below (See [Batch processing a \(sub-\)directory tree](#)).

For more on UTF handling see the other IM examples and information in [Unicode or UTF8 Format Text Handling](#).

Installing ImageMagick under Windows

ImageMagick is under constant development, new versions are released roughly on a monthly basis. It is strongly recommended to use an up-to-date version of IM, especially when IM doesn't seem to perform a job quite as you expect it to do. In most cases, the installation of the current version will solve the problem. The setup

program of the current binary release can be found at <http://www.imagemagick.org/script/binary-releases.php#windows>.

You can also download HDRI (Floating Point Quality) and FFT (Fast Fourier Transform) capable version of ImageMagick from [Astronomy and Astrophotography Blog](#).

By default, IM installs itself in a program directory called C:\Program Files\ImageMagick x.x.x-x, where "x.x.x-x" stands for its version number. By default, the setup program suggest to extend the PATH environment variable when IM is installed for the first time (i.e. "Add application path to your system path" is checked). If you forget to de-install older versions, you will quickly have a nice collection of various ImageMagick versions with their corresponding PATH extensions.

In our office, we therefore took the habit to install IM into C:\Program Files\ImageMagick, installing newer versions just over the older ones and leaving the PATH environment variable untouched. We couldn't find anything wrong with this way of proceeding in years of usage. If you really want to know the IM version number, you can always call "convert -version" and a dummy-proof script can evaluate the version number in case that it relies on a certain minimum version number. See section "[Editing, Debugging and Runtime Error Testing](#)" for more information on this.

ImageMagick writes a few Registry keys to HKEY Local Machine\Software\ImageMagick. If you do deal with several installed versions of IM, the most important key is HKEY Local Machine\Software\ImageMagick\Current, where you can also find the path to IM's binaries, called BinPath. You can query this registry entry at the start of any script and thus determine the program path without having to rely on the PATH environment variable. See section "[Editing, Debugging and Runtime Error Testing](#)" for more information on this.

IM's setup program offers the option to install a COM+ object for ImageMagick by the option *Install ImageMagickObject OLE Control for VBscript, Visual Basic, and WSH*. The option is not checked by default and the installation of the COM+ object is **no prerequisite** to use IM in a VBScript, as will be proven in the following. Anyway, you should not rely on the COM+ object to be installed on a script target machine other than your own.

When working with PostScript files, ImageMagick relies on another program, "Ghostscript" for the reading and conversion of PostScript and PDF files into a image format it can use. That is to read such documents, Ghostscript needs to be installed on your computer. Its latest version can be downloaded at [SourceForge](#).

The order in which you install GhostScript and ImageMagick does not matter. You do not have to install GhostScript prior to ImageMagick, and ImageMagick will run fine without it being installed. It is only needed if you want to work with Postscript or PDF files. IM will determines the location of Ghostscript at runtime, querying it from the Registry.

Specialities and pitfalls

Quite extraordinarily for Windows programs, IM allows images to be written to *stdout* and read from *stdin* and thus use piping to cascade image processing tasks. The operation

```
convert -size 128x128 xc:white gif:- | convert - test.gif
```

is equivalent to

```
convert -size 128x128 xc:white test.gif
```

In the first command of the pipe, the minus operator tells IM to write the image to *stdout*, while the minus operator in the second command of the pipe tells IM to read the image from *stdin*. This was of proceeding allows to avoid the explicit use temporary files. It is especially useful if some command does not offer certain operations, for example trimming:

```
montage -tile 2x2 -geometry 400x300+60+60 1.png 2.png 3.png 4.png miff:- | convert - -trim montage.png
```

As a consequence of this feature, textual output is usually written to *stderr* instead of *stdout*. For example: if you want to redirect the textual output of Compare to a text file, you would have to write

```
compare -metric PSNR 1.png 2.png dif_1_2.png 2>result.txt
```

So you have to redirect *stderr* ("2>") to the text file, not *stdout* ("1>" or just ">").

Getting Help

The command line options are extensively documented on [IM's website](#), but it's the [Usage Section](#) of its website which really demonstrates how to get them to work. This section of the website is quite well structured, allowing a problem-oriented approach to the task you would like to perform. However the quick-and-dirty approach is a Google search on that section of the website in regard to the command line option that you have in mind. If you want to perform a montage of several images and want to find out about the *-tile* option you could perform a Google search on either

- [Imagemagick Usage Tile](#)
- [tile site:www.imagemagick.org/Usage](http://www.imagemagick.org/Usage)

and you will quickly find out about the essentials. Just keep in mind that this will reveal code intended for application in a UNIX / LINUX environment, which has to be slightly adjusted in order to be applied under Windows.

Another very helpful source of information is the IM discussion board, aka the [Discourse Server User Forum](#), which you should incorporate in the bookmarks / favourites of your browser. Becoming a member will allow you to pose questions, which are mostly answered quickly by knowledgeable users.

Auxiliary programs and alternatives

Yes, it's true. There **are** certain jobs that other programs can do better than ImageMagick. Typically ones which are designed with specific formats in mind, rather than general image manipulation that ImageMagick provides.

- [IrfanView](#) is probably the most common image viewer under Windows, which also allows some basic image manipulation.
- A GUI program like Adobe Photoshop or [Gimp](#) is more suited for direct editing and testing complex image manipulation steps.
- For manipulations on the EXIF header of digital photographs, [Jhead](#) and [ExifTool](#) are more versatile than ImageMagick.

- Extraction of JPEG streams from PDFs should be done with [xPDF](#).
- Video processing should better be done with [VirtualDub](#), best of all in combination with [AviSynth](#) and its editor [AvsP](#).

This is not to say ImageMagick should be ignored for such image work. But you can do a lot more by combining them together.

Using Cygwin

As has been said above, ImageMagick was designed with UNIX and Linux in mind, so the easiest approach is probably to install a Bash shell on your Windows system and run the variety of IM scripts which have already been written for that system, for example [Fred Weinhaus' scripts](#).

[Cygwin](#) is – as its developers put it, provides a "a Linux-like environment for Windows". It consists of all the tools which are normally available in the Linux shell. I have tested a few of Fred Weinhaus' bash scripts under Cygwin's Bash shell and found them to be fully functional.

At the bottom of the root page of the Cygwin website, you will find a link labeled [Install or update now!](#), which will download an installation stub named Setup.exe. When you start this program, it will offer a list of site mirrors. After you have chosen one of them, the routine will provide a tree view of what tools it is going to install. The standard selection seems reasonable to me, so you might just proceed. The installation routine will then download the packages needed and install Cygwin on your computer.

When you start Cygwin, its Bash shell looks pretty much like a DOS box, that is a text window with black background. Like for the DOS window, the font can be chosen by clicking on the window's system menu, located left in the title bar. The basic commands are:

- You change the current directory with `cd` command, more or less like in DOS. However, the backslash ("`\`") has to be substituted by the forward slash ("`/`"). Drives are also changed via `cd`, as under DOS. As such `cd w:` will switch to drive `w:`. Unlike in a normal Unix environment, pathnames are case-insensitive. Special characters, such as umlauts, can be used. For some commands drivenames have to be passed in a special syntax, avoiding the colon. For example `/cygdrive/w/test`.
- Cygwin reads the Windows `PATH` environment variable and sets its own `PATH` correspondingly. You can check that by typing `echo $PATH` into the Bash shell. Note: Unlike under Windows, the names of environment variables are case-sensitive, so you have to use capital letters when referring to `PATH`.
- Unlike under Windows, the current directory is **not** in the search path by default! If for example the shell script "autolevel1" resides in `w:\scripts`, you can not just switch to that directory when calling the shell script. You must append at least a minimal directory location to the start of the script, like this: `./autolevel1` (for the script located in current directory). Or the script complete path, like this `/cygdrive/w/scripts`.
- Alternatively you can append that script directory explicitly to the search path using `PATH=$PATH:/cygdrive/w/scripts`. As the colon is used as a path separator, you have to use this special nomenclature, such as `/cygdrive/w/scripts` instead of `w:/scripts`.

This description of the Cygwin shell will be extended in future versions of this page. For the moment, the above information should suffice to get you started.

Using the DOS Shell and Batch Files

Converting Scripts: UNIX Shell to Window DOS

When invoking IM commands directly from the DOS command shell (using `cmd.exe`) you have to modify the scripts presented on IM's Example site (if they don't stem from this section), as most examples provided (in other sections) are generally intended to be run in a UNIX or LINUX command shell.

In order to run them from a DOS command shell, you have to perform the following modifications:

- Most often, double quotes "`"`" have to be used in place of single quotes "`'`" so that the command arguments remain correct. Watch out for quotes within quotes such as in [_draw](#) operator. You can use single quotes within a DOS double quoted argument as these are passed to IM for handling, and not processed by the script.
- Backslashes "`\`" appearing on the end of the shown example lines represents a 'line continuation' which appends the next line to the same command line sequence.

Replace them with "`^`" character to denote line continuation in DOS batch files. For DOS the next line will also need to start with a space, however that is fairly standard practice so should not cause much of a problem.

You can also append all the lines onto one line and remove those backslashes, though this makes editing, and later reading the command much harder. As such the practice is not recommended.

- All reserved shell characters which are not in double quotes must be escaped with a "`^`" (caret or circumflex) if used in a literal sense (i.e. not fulfilling their usual purpose). These reserved shell characters are: `&`, `|`, `(`, `)`, `<`, `>`, `^`.

This especially means that:

- The special character "`>`" (used for resize geometry) needs to be escaped using "`^`". For example `"-resize 100x100^>"`
- Similarly the 'internal fit resize' flag "`^`" needs to be doubled to become "`^^`".

- UNIX shell escaping backslashes "`\`" are not needed to escape parenthesis "`(`" or exclamation marks "`!`".
- Otherwise the UNIX shell escaping backslashes "`\`" will need to be replaced with a circumflex "`^`" character, when that escape characters such as "`<`" and "`>`". For example: `"-resize 100x100\>"` will become `"-resize 100x100^>"`.
- In DOS batch files, the percent "`%`" character also has a special meaning as it references to the command line parameters. For example `%1`, `%2`, ... (in UNIX shell a "`$`" sign is used for the same general meaning). In a DOS batch file, single percent signs (as they appear in the "`FOR` command") will need to be doubled to "`%%`".

ImageMagick itself generally only looks to see if a percentage sign is present, and does not care if more than one has been given. So unless it is part of a text label or comment string, doubling all percent signs generally does not hurt.

- Keep in mind that Windows filenames can include space characters. Spaces can also be used under UNIX but not so common. Such filenames which may contain spaces have to be included in double quotes "`file name.jpg`" or "`file name`".`.jpg`.

A filename passed to a script or batchfile via Drag & Drop or SendTo as a command line parameter needs special attention, as it is passed to the script without quotes, if it doesn't contain space characters, and with quotes when it does.

- Comments in UNIX shell scripts start with an unquoted '#' anywhere in a line and continue to the end of the line. Color settings (such as "#FF0000" for a red color) will often be quoted with to remove this special meaning. This quoting is not needed, but using double quotes "" around them does not matter and should be kept.

In DOS, comments can only appear at the start of a line using a 'REM', '@REM', or ':' prefix. Though they also continue to the end of the line. It is your choice which method of commenting you should use. However good commenting in any batch file is always recommended, so you know what the command is attempting to do at each step, when you go back to the script months or years later. Makes it a lot easier for others to figure out too.

All scripts should start with a larger comment, explaining what the script does and how it should be used. This is just good programming practice.

- When executing a DOS batch file, the individual commands are echoed by default, That is commands are displayed in the output DOS box. Under UNIX you would instead need to add a special command or option to print commands being executed in this way.

You can turn off this 'echoed' output by starting your script with "@ECHO OFF".

The special starting comment "#!/path/to/shell" in UNIX shell scripts is not needed for DOS batch files. So this line can be replaced by the "@ECHO OFF" command for DOS batch files.

For example, following the above rules, this UNIX shell script...

```
#!/bin/sh
# Create a negated rose image and overlay a comment
convert -background none -fill red -gravity center \
  -font Candice -size 140x92 caption:'A Rose by any Name' \
  \( rose: -negate -resize 200% \) +swap -composite \
  output.gif
```

will become something like this as a Windows DOS batch file...

```
@ECHO OFF
:: Create a negated rose image and overlay a comment
convert -background none -fill red -gravity center ^
  -font "C:\path\to the\font\candice.ttf" ^
  -size 140x92 caption:"A Rose by any Name" ^
  ( rose: -negate -resize 200% ) +swap -composite ^
  C:\where\to\save\output.gif
```

I have written a basic Linux shell to DOS batch file converter by the use of SED (Streaming EDitor). SED is a common UNIX / Linux text file manipulation program which is also available for Windows at <http://sed.sourceforge.net/>. Like IM is a command-driven image manipulator, SED is a command-driven editor.

The SED script cim.txt that performs the needed manipulations looks like this (when stripped of any comments):

```
s/'/\\"/g
s/%/%/g
s/\\([()!])\\1/g
s/[&<>]\/^1/g
s/^[ ]*#/:/
s/(^.*\\)( #.*$)\/2\n1/
s/(\\..*\\.[a-z,A-Z]*\\)[ ]\/1\" /g
s/[ ]*$/^/
s/^[ ][ ]//
```

You can download the fully commented version file [sed_script.zip](#).

If you place the SED script cim.txt in the same folder as the Linux shell script which is to be converted, you invoke the conversion by:

```
%programfiles%\GnuWin32\bin\SED -f cim.txt linux.scr > windows.bat
```

You can also invoke the conversion via SendTo or Drag & Drop by use of the following batch file:

```
SET SP=%programfiles%\GnuWin32\bin
%SP%\SED -f %SP%\cim.txt "%~1" > "%~dpn1.bat"
```

This batch file assumes that you have placed the SED script cim.txt within SED's program folder. It takes the filename of the Linux shell script as the only command line parameter and generates a batch file with the same name, but extension '.bat' in the same folder. (The cryptic filename manipulation "%~dpn1.bat" is explained in the next section.)

Please note: The above SED script will only perform the rudimentary replacements mentioned above. It will NOT turn sophisticated Linux shell scripts (like those presented on [Fred Weinhaus' website](#)) into the equivalent batch file!

Filename Handling in Batch files

As has been said above, IM is particularly useful when applying a standard sequence of processing steps to an image file. In such a case, the filename will be passed to the script as a command line parameter, either via Drag & Drop or via SendTo. Using these techniques, the filename handed to the DOS batch file will be a fully qualified filename, i.e. include the drive name and the directory path. You can test this by dropping a file onto the following batch file:

```
ECHO Filename: %1
PAUSE
```

Due to the `PAUSE` statement, the DOS box will stay open until the user presses a key, such that you can inspect the result. Try the above with a filename that contains spaces and you will notice that the filename will be bracketed by double quotes.

Please note that here and in all following examples we assume that any network file has been assigned with a drive letter. Practically speaking, I have never seen different in a local commercial network. When working with batch files, you should not try to handle UNC names: You may get your batch working, but it will cause you a lot of unnecessary trouble.

When using this filename in an `Convert` command line, this behaviour can cause trouble. Let us perform a simple conversion from any other format to JPEG. The most basic code would be:

```
convert %1 %1.jpg
PAUSE
```

This will produce a JPEG file (with standard quality and resolution) in the same directory, tailed with an additional ".jpg" extension. The above code works on any filename, whether it contains spaces or not. If you want to get rid of the original extension, things become a little trickier:

```
convert %1 "%~dpn1.jpg"
PAUSE
```

The above batch file manipulates the filename by use of the `~` operator:

```
%~l   expands %l removing any surrounding quotes ("")
%~fl  expands %l to a fully qualified path name
%~dl  expands %l to a drive letter only
%~pl  expands %l to a path only
%~nl  expands %l to a file name only
%~xl  expands %l to a file extension only
```

These modifiers can be combined, such that "%~dpn1" means "drive + path + name without extension and bracketing quotes". Consequently, we have to bracket the name by double quotes, such that the code also works for filenames including spaces. (If it doesn't, the quotes do no harm.) The `PAUSE` statement is for testing purposes only and can be dropped in the final batch file.

If you just want to test your code without actually invoking IM, you should write:

```
ECHO convert %1 "%~dpn1.jpg"
PAUSE
```

which will just show the result of your string manipulation.

Batch Processing Several Files

The DOS `FOR` command can be used to process a series of files in a similar manner as it does under UNIX. In order to scale all JPEG files in the current directory by 50%, you could type the following line into a DOS box:

```
FOR %a in (*.jpg) DO convert %a -resize 50% small_%a
```

Please note that the percent sign is **not** doubled. If you however place this command in a batch file you will have to replace it by

```
FOR %%a in (*.jpg) DO convert %%a -resize 50%% small_%%a
```

Again, it is convenient to invoke this bulk operation by Drag & Drop or `SendTo`, passing a fully qualified filename (or a folder name) to a DOS batch file which is possibly located in another directory (such as `shell:sendto`). In this case, we should make the file's directory the current directory in the first step:

```
%~d1
CD "%~p1"
MD small
FOR %%a in (*.jpg) DO convert %%a -resize 50%% small\%%a
PAUSE
```

In this batch file we

- change the drive by supplying the drive name (d: or whatever)
- make the file's folder the current directory
- create a sub-directory named "small"
- scale all JPEG files by 50% and place these shrunken versions in the new sub-directory.

Please note: As the tilde (`~`) frees the filename passed as a command line parameter of any possibly bracketing quotes, we usually have to bracket the result of any such manipulation by quotes again. We therefore wrote `CD "%~p1"` in the example above. In case of the `CD` command, we might even have omitted the bracketing quotes, as this very command accepts only one parameter and can therefore handle blanks without bracketing quotes.

Making the file's directory the current directory in the first step does make the follow-up steps a little easier, as the references to filenames become a little easier and shorter. We could however as well have written:

```
MD "%~dp1small"
FOR %%a in ("%~dp1*.jpg") DO convert "%a" -resize 50%% "%~dp1small\%%~nxa"
PAUSE
```

This might make things a little shorter, but also a lot trickier. Please note that the file name modifiers also work on the For loop variable %%a. Note two: The final backslash is part of the pathname. Therefore it must be "%~dp1small" and **not** "%~dp1\small", which doesn't make the code more readable, especially in the case of "%~dp1small\%%~nxa".

There are several shortcomings and caveats of the FOR statement.

One of them is that you basically perform only one single command after DO. You can however group a series of DOS commands in parentheses "(" , ")" and thereby perform a simple sequence of commands:

```
@ECHO OFF
:: Lighten darker areas of all images in a directory
%~d1
CD "%~p1"
FOR %%a in (*.jpg) DO (
    ECHO Processing file: "%~nxa"
    convert %%a -blur 30 -negate %%a.miff
    composite %%a.miff %%a -compose overlay "%~dpn1_light"%%~xa
    DEL %%a.miff
)
PAUSE
```

This batch file will process all images found in the directory passed as the command line argument. First it blurs the original image and negates it, storing the intermediate result in a file with an additional ".miff" extension. Then it superposes the original image over this modified version, thereby lightening the darker sections of the original image. Finally the intermediate image is deleted.

Please note that in the above, emphasis must be put on the **simple sequence of commands**: You cannot make use of GOTO jumps within the block. If you need such behaviour, you have to call another batch file by the FOR loop:

```
%~d1
CD %~p1
MD small
FOR %%a in (*.jpg) DO CALL "%~dp0process" "%~fa"
```

Where "process.bat" is the batch file which does the actual work and which is located in the same directory as the calling batch file. The command line parameter 0 ("%0") is the name of the batch file itself, such that "%~dp0process" calls the batch file process.bat in the same directory. The FOR statement provides just the filename, which is turned into a fully qualified filename via "%~fa". In the present case, the code in the batch file process.bat would be the same as the one that was put between the parantheses in the above example:

```
convert %1 -blur 30 -negate %1.miff
composite %1.miff %1 -compose overlay "%~dpn1_light"%%~x1
DEL %1.miff
```

The use of a separate batch file, however, offers all the (limited) possibilities of a DOS batch file. This does not make any difference in this example, but we will show the benefits of this approach further down.

If you don't want to bother with two batch files, you can have the script create the second process.bat script (using ECHO), call it from the main loop, then deleted it when the job is finished:

```
ECHO convert %1 -blur 30 -negate %1.miff >%~dp0process.bat
ECHO composite %1.miff %1 -compose overlay "%~dpn1_light"%%~x1 >>%~dp0process.bat
ECHO DEL %1.miff >>%~dp0process.bat
%~d1
CD %~p1
MD small
FOR %%a in (*.jpg) DO CALL "%~dp0process" "%~fa"
DEL %~dp0process.bat
```

When using the ECHO command, we have to escape any special DOS characters, especially the percent sign, a second time.

And yes, IM could have done all the above in a single processing command, removing the need for the ".miff" intermediate image, but that is not the point of this example.

Batch processing a (sub-)directory tree

There are several techniques to process all files in a (sub-)directory tree. The simplest approach is to use the "/R" flag in the FOR statement to make it loop over all the files in all sub-directories under the current directory. In order to convert all TIFF files in the subdirectory tree to JPEG you thus simply type:

```
FOR /R %%a IN (*.tif) DO imconv "%~a" "%~dpna.jpg"
```

When using the "/R" flag, you are always looping through the entire subdirectory tree, without options for sorting or filtering files.

In the oncoming example, we will generate photo index prints for all subdirectories and place these within the root directory. This offers an easy way to perform a visual search for a certain photograph, similar to the preview in the Windows Explorer, but without the (time-consuming) need to re-scan the entire directory tree for each

search.

As a start, we approach the problem with the help of two batch files, one performing the loop and one doing the actual work. The index prints will be low-quality JPEG files named `IDX_0001.jpg`, `IDX_0002.jpg`, `IDX_0003.jpg` and so on. First we establish the loop:

```
DEL IDX_?????.JPG
SET COUNT=0
FOR /F "delims=" %a in ('DIR /S /B /AD ^|FIND /I "Porsche" ^|SORT') DO CALL c.bat "%a"
DEL title.txt
```

The first line cleans up any results from previous searches. In the second line, we define the environment variable `COUNT`, which we will use to generate the `IDX_nnnn.JPG` filenames. In the third line, we establish a list of all subdirectories via `DIR /S /B /AD`, extract those directories that contain the word "Porsche" (case-insensitive by use of the option `/I`) and sort this filtered list. Sorting will ensure that the numerical ordering of the `IDX` files will coincide with the alphabetical ordering of the directory pathnames. The option `"delims="` will inhibit the standard behaviour of truncating the lines after the first blank.

When calling the batch file `c.BAT`, we bracket the pathname by quotes to ensure that blanks are treated correctly. In the last line, we delete a temporary file that is created by the batch file `c.BAT`. We now come to the actual work:

```
CHCP 1252
DIR %1\*.jpg>nul || GOTO :EOF

:: Generate IDX filename
SET /A COUNT+=1
SET TFILE=000!COUNT%
SET TFILE=IDX_%TFILE:~-4%.jpg

:: Generate title without bracketing quotes
ECHO %~1>temp.txt
"C:\Program Files\Gnuwin32\bin\iconv.exe" -f ISO-8859-1 -t UTF-8 temp.txt>title.txt

montage -geometry 210x140+0+5 -tile 6x -title @title.txt %1\*.jpg -quality 30% %TFILE%
jhead -c1 %1 %TFILE%
```

In the first line we switch the codepage to Western European Latin (ISO-8859-1). In the second line, we check whether the directory actually contains any photographs and skip the rest of the batch if not. (Executing the rest of the batch instead would actually do no harm, as `montage` would simply generate no output, but the count of the `IDX` files would no longer be consecutive.) Since Windows XP there is a special `GoTo` target label `:EOF` which allows to terminate the execution without defining an adequate label.

We then generate the filename `"TFILE"` of the index file by incrementing `"COUNT"`, attaching some leading zeros and extracting the last 4 characters via `%TFILE:~-4%` then concatenating it to create a filename of the form `"IDX_nnnn.jpg"`. The use of the `SET /A` statement to perform calculations is explained in [Calculations using SET](#) later.

In the following lines we free the pathname `"PNAME"` of the quotes and store the result in the intermediate file `'temp.txt'`, which is transcoded to Unicode (UTF-8) by the help of `"Iconv.exe"`, (see ["Character Encoding"](#)). The Unicode string stored in `'title.txt'` is then read by IM's `Montage`. This ensures that the string is treated literally, so that we don't have to escape the backslashes in Windows pathnames.

`Montage` then combines the photographs in rows by six (`-tile 6x`) and titles them with the pathname. The resulting index print will be 1260 pixels wide and is stored with 30% JPG quality in order to reduce storage demands.

In the last line, we use the program [JHEAD](#) to write the pathname into the JPEG comment. This offers the possibility to filter the index prints within the Windows Explorer by text-searching the files for certain substrings in the filename.

We can combine the two batch files, placing the code of the "working batch" into the `FOR` loop:

```
SETLOCAL EnableDelayedExpansion
SET ICONV="%PROGRAMFILES%\Gnuwin32\bin\iconv.exe" -f ISO-8859-1 -t UTF-8
CHCP 1252
DEL IDX_?????.JPG
SET COUNT=0
FOR /F "delims=" %a in ('DIR /S /B /AD ^|FIND /I "Porsche" ^|SORT') DO (
  DIR "%a\*.jpg">nul
  IF !ERRORLEVEL!==0 (
    SET /A COUNT+=1
    SET TFILE=000!COUNT!
    SET TFILE=IDX_%TFILE:~-4!.jpg
    ECHO %a >temp.txt
    %ICONV% temp.txt>title.txt
    montage -geometry 210x140+0+5 -tile 6x -title @title.txt "%a\*.jpg" -quality 30% !TFILE!
    jhead -c1 %a !TFILE!
  )
)
DEL temp.txt
DEL title.txt
```

Basically, two modifications have to be applied to the initial code:

- We have to enable **delayed expansion** and refer to the environment variables used within the `FOR` loop by bracketing them with exclamation marks instead of percent signs.
- We have to avoid the `GOTO` statement which would reset the command processor.

Per default, the environment variables within a `FOR` loop are **not** evaluated at runtime. Instead, the code is pre-processed using the list given in the parenthesis. A reference to `%COUNT%` within the `FOR` loop therefore always hands back the same value. In order to enable the runtime evaluation of environment variables, you have to switch on delayed expansion. This can be done when calling the command processor via `cmd /V:on` or be generally switched on in the registry, using the following REG file:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Command Processor]
"DelayedExpansion"="1"

[HKEY_CURRENT_USER\Software\Microsoft\Command Processor]
"DelayedExpansion"="1"
```

However, the dummy-proof version is setting this option in the batch itself by use of `SETLOCAL EnableDelayedExpansion` which will also limit any changes to environment variables to the current version of the command processor, which is probably desired. References to the runtime values of environment variables within the loop now have to use exclamation marks instead of percent signs.

The use of `GOTO` statements within a `FOR` loop is a possible source of very subtle errors and should therefore be avoided. In our batch we can however easily exchange the jump by an `IF` statement bracketing the code block with the montage code.

Batch Processing an Arbitrary Number of Files

In DOS batch files, only nine command line parameters can be addressed directly by %1 to %9. In former Windows versions, you could only circumvent this limitation by the `SHIFT` command, which caused a circular shift of the command line parameters. In newer versions, the command line parameters can be treated in a `For` loop:

```
FOR %i in (*) DO ...
```

This allows us to Montage an arbitrary number of images passed by Drag & Drop:

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion
%~d1
CD "%~p1"
DEL Files.txt 2>nul:
DEL FSorted.txt 2>nul:
FOR %I in (*) DO ECHO %~nxi>>files.txt
FOR /F "delims=" %A in ('TYPE files.txt ^| Sort') Do (
    ECHO %A>>fsorted.txt
)
SET MONTAGE=montage -tile 3x
FOR /F "delims=" %A in (FSorted.txt) Do (
    SET MONTAGE=!MONTAGE! %A
)
SET MONTAGE=%MONTAGE% result%~x1
%MONTAGE%
DEL Files.txt
DEL FSorted.txt
```

The above code assumes that the files handed over as command line parameters reside in the same directory and have to be montaged according to the alphanumerical order of their filenames. This would for example make sense for an index of digital photographs which would thereby be ordered according to their shooting time. We first make the files' directory the current directory, which makes the further code a lot simpler. We then dump the filenames into `files.txt`. Even if the files were picked in alphabetical order, we cannot rely on that this ordering is preserved when the filenames are handed over to the script. Therefore, we order the files in a second working step, dumping them in `fsorted.txt`.

Based on that file, we then construct the Montage command line in another `For` loop. The Montage output file uses the same extension as the first input file (%~x1). (Assuming, that all files share the same extension.) Please note that the final Montage command is just called by evaluating the environment variable, i. e. `%MONTAGE%`.

As has been said in , handing over a large amount of files with long (full) filename can get you into trouble under Windows XP, because the parameter list of `ShellExecute` is limited to 2048 characters. This error cannot be handled by the batch file, as it occurs **before** the control is handed over to the batch file. A possible script-based solution is to process all image files in the directory when just one file is handed to the batch:

```
%~d1 & IF EXIST %1\* (CD %1) ELSE CD %~p1
IF "%2"==" " (
    SET PATTERN=*.jpg
) ELSE (
    SET PATTERN=%*
)
FOR %%v IN (%PATTERN%) DO (...)
```

In the first line we check whether the second command line parameter is present. If so, all command line parameters (%) are processed. If only one file is handed to the script, the pattern is set to all JPEG files. This simple pattern match requires that we have change to the current drive (via %~d1) and the current folder, i.e.

- `CD %1` if a folder name was handed to the script
- `CD %p1` if a filename was handed to the script.

Re-using the Output of an IM command

In recent versions of Windows, the `FOR` statement has become much more powerful, see [DOS "For" Command Help](#). By using the `/F` option, you can read the input for the substitution variable from a file, a string or from the output of another DOS command or another program. The latter is especially useful with IM.

To get a rough idea on what IM's overlay methods really are about, you could use the following batch file:

```
@ECHO OFF
:: compose two gradients using all compose methods available
::
convert -size 80x80 -flip gradient: compose_src.png
```

```
convert compose_src.png -rotate 90 compose_dst.png
FOR /F %A in ('convert -list compose') DO ^
    composite compose_src.png compose_dst.png -compose %A compose_%A.png
```



Src



Dst



Multiply



Screen



Overlay

This script composes two gradient images together, using every possible [Alpha Composition method](#) available, so you can see how operators effect image colors. Only some of the images it generates are shown above. This is similar to the images that were generated for [Composition Tables](#), so you can see how operators affect image colors.

As the above lines are assumed to be a batch file, we have to double the percent signs.

The above script first creates two orthogonal (right angle aligned) gray-scale images with gradients covering the entire grey-scale range. The IM command `convert -list compose` will provide us with a list of possible options, each placed within a single output line. Please note that we have to use single quotes when referring to a command in the parenthesis.

Using the `/F` option, the `FOR` command will then process each of these output lines and hand it over to the command executed by `DO`. As a consequence, the two gradient images are superposed applying all the overlay methods that IM knows of. The output files are named correspondingly to the overlay method.

In the next example, we illustrate the color spaces which IM provides. We use the same gradient technique as above to generate the surfaces of a cube as spanned by the three coordinates of the colour space:

```
convert -size 256x256 gradient: gy.miff
convert gy.miff -rotate 90 gx.miff
convert -size 256x256 xc:black black.miff

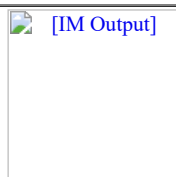
::R + G top left
convert gy.miff gx.miff -flop black.miff -set colorspace %1 -combine ^
    -resize 260x300! -background none -shear 0x-30 ^
    -virtual-pixel Transparent RG.miff

:: R + B top right
convert gy.miff black.miff gx.miff -set colorspace %1 -combine ^
    -resize 260x300! -background none -shear 0x30 RB.miff

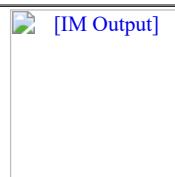
:: G + B bottom
convert black.miff gx.miff gy.miff -set colorspace %1 -combine ^
    -resize 260x300! -background none -shear 0x30 -rotate 120 ^
    -crop 520x300+0+75 GB.miff

convert -set colorspace %1 RG.miff RB.miff +append top.miff
convert -set colorspace %1 -size 520x150 xc:Transparent w.miff
convert top.miff w.miff -append topx.miff

composite -geometry +0+299 GB.miff topx.miff colorspace_%1.png
DEL *.miff
```



colorspace RGB



colorspace sRGB

A similar example to the above using UNIX shell scripting is given in [Isometric Cube using Shears](#).

This batch file takes the color space as a command line parameter `"%1"`. It then generates the three sides of the cube and shears and mounts them such that we get an isometric view, where the point (0,0,0) lies at the centre of a hexagon. The final picture is named after the color space (IE. `"%1"`) and stored as a PNG.

We now want to call this batch file (saved as `"cspace.bat"`) from another batch file that provides the names of the colour spaces:

```
FOR /F %A in ('convert -list colorspace') DO CALL cspace %A
```

We can also filter the output of the `-list` option by piping it in DOS:

```
convert -list colorspace | FIND "RGB" >>clist.txt
FOR /F %A in (clist.txt) DO CALL cspace %A
DEL clist.txt
```

In this example, we filter those lines from the output that contain `"RGB"` and write them to the file `clist.txt`. This file is then used as the input for the `FOR /F` command. We can also do this in one run, avoiding the temporary file:

```
FOR /F %A in ('convert -list colorspace ^| FIND "RGB"') DO CALL cspace %A
```

In this case, the pipe symbol "|" has to be escaped, because it is not bracketed by double quotes (only by the single quotes needed for the FOR statement) and is (at least in command line above) not meant in its usual sense.

Processing Single Line Output

This technique can also usefully be applied to a single-line output. We can for example apply an automatic gamma correction that roughly sets the average brightness of a picture to the middle of the quantum range (i.e. 127 for a color depth of 8 bit) by a technique explained on [Fred Weinhaus' website](#):

```
FOR /F %a IN ('identify -format "%[fx:log(mean)/log(0.5)]" %1') DO ^
convert %1 -gamma %a "%~dpn1".c.%~x1
```

This batch file is handed a fully qualified filename as the command line parameter "%1", most likely via Drag & Drop or SendTo. The output of IM's Identify command then provides with a gamma value, which will set the image's average brightness to the middle of its dynamic range. This value is calculated using a [FX Format Expression](#). The single line output of the Identify command is saved into the "%a" variable, and passed to the Convert command as an argument for the [Gamma Operator](#).

Please note that the FOR command seems to be quite sensitive when it comes to line continuations: If you use them at all, make sure that you better don't start the next line with spaces.



This method of automatic gamma correction is now built into IM using ["-auto-gamma"](#), and was added to IM v6.5.5-1. But it shows the technique of re-using command output for use in later command arguments.

With the same FOR technique, we can read from the EXIF information embedded in a photograph and write it into the top left corner of the image:

```
FOR /F "tokens=1,2" %i IN ('identify -format "%[EXIF:DateTime]" %1') DO ^
convert %1 -pointsize 18 -fill white -gravity northwest ^
-annotate +0+0 "%i %j" "%~dpn1".dated.%~x1
```



Photos are typically saved using JPEG format. Reading and re-saving JPEG images causes slight degrading of the image due to [JPEG Lossy Compression](#) and as such saving back to JPEG is not recommended.

In the above batch file, the filename of the photograph is supplied as the single command line parameter, which is referred to as "%1". The Identify command reads the date and time that the photograph was taken from the EXIF information within the JPEG file. The FOR command then hands this output over to Convert which annotates the photograph correspondingly in the upper left corner.

The EXIF date and time information is formatted as "yyyy:mm:dd hh:mm:ss", e.g. "2006:12:26 00:22:38". Thus date and time are separated by a space character. By default, the FOR statement would only find the first token ("word") in each line, with tab and space characters as the standard delimiters. Thus in the example above, the standard processing would only handle back the date, but not the time. Using the option "tokens=1,2" we declare our interest in both tokens, which are named consecutive, i.e. "%x, %y". We can however change the rather unconventional formatting of the date by the following code:

```
FOR /F "tokens=1,2,3,* delims=: " %i IN ('identify -format "%[EXIF:DateTime]" %1') DO ^
convert %1 -pointsize 18 -fill white -gravity northwest ^
-annotate +0+0 "%j/%k/%i %1" "%~dpn1".dated.%~x1
```

We now defined the colon (":") as an additional delimiter, causing the date to be broken up into the three tokens "%i", "%j", "%k". The next delimiter found is the space character separating date and time. With the asterisk ("*") we asking the rest of the line to be stored in the fourth token "%1". We can now format the date as we like to. We have chosen the Anglo-American notation "mm/dd/yyyy" as in the above example.

Performing Calculations

The DOS command interpreter is poor when it comes to calculations. You can use it to perform simple integer arithmetics. But for doing more complex floating point mathematics, you have access to IM's [FX Format Expression](#), or a third party DOS calculator program.

Using IM's FX Expressions

IM's [FX Format Expression](#) can be used for floating point mathematics and can add that maths to larger formatted strings, as has been demonstrated above in the first example of the section [Processing Single Line Output](#). By use of the SET command, the result can be stored in an environment variable and used later in the batch file. As a simple example, we may wish to adjust the font size date-time string in the above example corresponding to the dimensions of the photograph:

```
FOR /F %i IN ('identify -format "%[fx:min(w,h)*0.05]" %1') DO SET psize=%i
FOR /F "tokens=1,2,3,* delims=: " %i IN ('identify -format "%[EXIF:DateTime]" %1') DO ^
convert %1 -pointsize %psize% -fill white -gravity northwest ^
-annotate +0+0 "%j/%k/%i %1" "%~dpn1".dated.%~x1
```

In the first line we evaluate the smaller dimension of the photograph by "%[fx:min(w,h)]", take 5% of this value and store it in the environment variable PSIZE. This value is referred to in the next statement (%psize%) to set the font size of the time-date information.

And here we calculate a random angle between as an integer between -15° and +15° to create a rotated thumbnail image.

```
FOR /F %x IN ('convert null: -format "%[fx:int(rand()*31)-15]" info:') DO SET angle=%x
convert %1 -thumbnail x90 -matte ^
-background none -rotate %angle% "%~dpn1".rotated.png
```

The [FX Expressions](#) cannot only generate numbers, but can also generate multiple numbers, embedded in a larger string. For example in [Border with Rounded Corner](#) it was used to directly generate a complex draw string based on image width and height information. This added feature, along with avoiding and further dependence on other external programs makes this method a preferable method for doing calculations in your batch script.

Using the SET command

The SET command can perform some simple integer math and some basic string manipulation when the "/A" option is invoked. In the following example, we roughly calculate the width of the time-date string by use of the SET command:

```
:: Determine the font height
FOR /F %i IN ('identify -format "%[fx:min(w,h)*0.05]" %1') DO SET psize=%i

:: The width of the date-time string is roughly 9 times its height
SET /A pwidth=%psize * 9

:: Calculate the average brightness in this section
:: and choose the text color correspondingly
FOR /F %i IN ('identify -format "%[fx:mean]" -crop %pwidth%x%psize%+0+0 %1') DO SET mean=%i
IF %mean% LEQ 0.5 (SET fcolor=white) ELSE SET fcolor=black

:: Annotate the photograph
FOR /F "tokens=1,2,3,* delims=" %i IN ('identify -format "%[EXIF:DateTime]" %1') DO ^
convert %1 -pointsize %psize% -fill %fcolor% -gravity northwest ^
-annotate +0+0 "%j/%k/%i %1" "%~dpn1" _dated%~x1
```

This sample batch file chooses the color of the date-time string corresponding to the average brightness (%mean%) in the area where it will be placed. If the average color intensity is less than 50%, the string will be white, otherwise it will be black.

The example also makes use of the IF statement. Please note that the ELSE part has to be placed in the same line and that the first command has to be bracketed.

Using Other External Calculators

As an alternative, you can use a DOS program which provides floating point math, such as [EVAL](#). If you place this file in the IM program directory or in the Windows system directory, you can perform floating point calculations in any DOS shell window.

By using the EVAL program, the FOR command and environment variables, we can make the color cube example from above somewhat more flexible and its various calculations more transparent:

```
convert -size 256x256 gradient: gy.gif
convert gy.gif -rotate 90 gx.gif
convert -size 256x256 xc:black black.gif

:: Set the dimension of the color cube / hexagon and calculate the various lengths
SET l1=512
FOR /F %i IN ('EVAL "round(cos(degree*30)*%l1%)"') DO SET l2=%i
FOR /F %i IN ('EVAL "2*%l2%"') DO SET l3=%i
FOR /F %i IN ('EVAL "round((1+sin(degree*30))*%l1%/2)"') DO SET l4=%i
FOR /F %i IN ('EVAL "round(%l1%/4)"') DO SET l5=%i

convert gy.gif gx.gif -flop black.gif -set colorspace %1 -combine ^
-resize %l2%x%l1%! -background none -shear 0x-30 ^
-virtual-pixel Transparent RG.miff

convert gy.gif black.gif gx.gif -set colorspace %1 -combine ^
-resize %l2%x%l1%! -background none -shear 0x30 RB.miff

convert black.gif gx.gif gy.gif -set colorspace %1 -combine ^
-resize %l2%x%l1%! -background none -shear 0x30 -rotate 120 ^
-crop %l3%x%l1%+0+0%l5% GB.miff

convert -set colorspace %1 RG.miff RB.miff +append top.miff
convert -set colorspace %1 -size %l3%x%l4% xc:Transparent w.miff
convert top.miff w.miff -append topx.miff

composite -geometry +0+%l1% GB.miff topx.miff colorspace_%1.png
DEL *.miff
DEL *.gif
```

Editing, Debugging and Runtime Error Testing

In principle, DOS batch files can be written in any editor, even with Windows' Notepad. You should however use an editor with syntax highlighting for DOS batch files. I personally think that [Notepad++](#) is the tool of choice, but talking of editors tends to make people nasty. So: yes, any other editor will do.

As far as I know, there is no free batch file IDE (integrated development environment) on the market. One would think that this should be something which comes with the operating system, but this has never been the case. So far, I have written all my batch files with Notepad++, but for those who write batch files on a regular basis, the [Running Steps](#) batch IDE might be of help. It is shareware and costs about \$80.

Comprehensive explanations of the DOS commands can be found at <http://www.computerhope.com/msdos.htm>.

As the DOS batch language itself, debugging batch files is a rather odd business. I would test any batch file in a DOS box as a start. When testing Drag & Drop or SendTo, it is recommendable to end the batch file with a PAUSE statement such that the DOS box stays open after the batch job has been finished.

Considering runtime error messages, the general approach is to check the DOS ERRORLEVEL and jump to an corresponding error message generated by the ECHO command. I found that one of the most probable error sources is that the Convert programm is not properly found on the machine running the script. So if you intend to share your batch scripts with others, you should first of all check whether Convert is accessible:

```
@ECHO OFF
convert -version 1>nul: 2>nul:
```

```
IF NOT %errorlevel%==0 GOTO NoMagick:
convert ...
GOTO :EOF
...
:NoMagick
ECHO ImageMagick (convert.exe) not found.
PAUSE
```

In the first line, we question the ImageMagick version, suppressing the standard output by `>nul:` (or just `>nul:`) and any error message by `>&nul:`, i. e. we redirect `stdout` and `stderr` to `nul:`. If the call to IM's `Convert` fails, the system program `Convert` will be called instead, which cannot handle the `-version` option and will set the `ERRORLEVEL` variable.

You might try to determine why `Convert` is not found and attempt to fix the problem: You can determine whether IM's program path is part of the environment variable `PATH`:

```
@ECHO OFF
PATH | FIND /I "ImageMagick"
IF NOT %errorlevel%==0 GOTO NoPath:
...
:NoPath
...
```

If `Find` (called with the case-insensitive option `/I`) cannot find the string, it sets the `ERRORLEVEL`. In a more sophisticated approach, you can check the Registry entry instead, no longer relying on `PATH`:

```
@ECHO OFF
FOR /F "tokens=1,2,*" %%A in ^
('reg query "HKCM\Software\ImageMagick\Current" ^| FIND "BinPath"') DO ^
SET MPATH=%%C
IF [%MPATH%]==[] GOTO NoMagick:
"MPATH\convert.exe" ...
...
:NoMagick
...
```

With this code, we query IM's Registry key `Current` and search for the entry `BinPath`. The decisive line of the output is:

```
LibPath REG_SZ C:\Program Files\ImageMagick
```

The "words" in this line of text are separated by tabs (in Windows XP) or several blanks (Windows Vista). These are the standard delimiters used by `For /F`. The third "word" (`%%C`) is the one we are looking for and we store it in the environment variable `MPATH`, which we can refer to when calling `convert` later in the script.

A script might require a certain minimum version number of ImageMagick to be installed. For example, the [Perspective Distortion Method](#) was first implemented in version 6.3.5-9 (in September 2007). So if your script deals with perspective rectification, you should test whether the installed version of IM is newer than that:

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion
SET MINVERSION=7.5.3-0
FOR /F "tokens=1,2,3" %%a in ('convert -version ^|FIND "Version"') DO SET VERSION=%%c
IF %VERSION% LSS %MINVERSION% GOTO GetNewVersion:
Goto :EOF
:GetNewVersion
ECHO This Script requires et least ImageMagick version %MINVERSION%.
ECHO Yours is %VERSION%.
PAUSE
Goto :EOF
```

`SETLOCAL` restricts any changes to environment variables to the current script, such that we do not have to fear side effects. The option `EnableDelayedExpansion` is not really needed over here, but it's good habit to use that option anytime you use `SETLOCAL`. We then store the minimum required version in the environment variable `MINVERSION`. In the third line, we call `Convert` with the `'-version'` option, extract the first line from the output via `^|FIND "Version"`, get the third word from that line and store it in the environment variable `VERSION`. We then compare this version to the minimal required version in the forth line.

Optimising execution time

To measure the execution time, you can display the content of the `%TEMP%` environment variable. The following script tests various ways to calculate the average brightness of a (large) image file, say a digital photograph:

```
IF "%1"==" " GOTO :EOF
ECHO %TIME%
Identify -verbose %1 | FIND "mean" & ECHO %TIME%
copy %1 %TEMP%\*. * & ECHO %TIME%
Identify -verbose %TEMP%\%1 | FIND "mean" & ECHO %TIME%
Convert %TEMP%\%1 -format %[fx:mean] info: & ECHO %TIME%
Convert %TEMP%\%1 -resize 1x1! -format %[fx:mean] info: & ECHO %TIME%
DEL %TEMP%\%1
```

The image is handed to the script as the single command line parameter, which is tested in the first line of the script. In the following, the execution time of various approaches is measured by echoing the environment variable `%TIME%` before and after the statement. The ampersand `&` allows to put several DOS commands in one line and is just used to shorten the space needed for the code over here. If the image is placed on a network drive, the transfer to the memory of the client computer may consume a considerable amount of the execution time. Therefore the file is copied to the local temporary folder, whose name is stored in the environment variable `%TEMP%`.

It turns out that the simply `Convert` command takes just as much execution time as filtering the result of `Identify`, but that resizing the image to one pixel via `-resize 1x1!` significantly fastens the operation, without changing its result too much.

Note that we are using `ECHO %TIME%` instead of `TIME /T`, because the latter will only display hours and minutes, whereas the former will provide hundredths of a second. In contrast to UNIX command shells, there is no direct way to measure relative times, i.e. set a reference point, execute the statement, and then evaluate the time needed. The calculation of relative times within a batch file is made difficult by the fact that batch files allow only integer arithmetics. One can extract the seconds by the `FOR` command and then use IM's `fx` operator to perform the subtraction, allowing for 60 seconds overflow

```
FOR /F "tokens=1,2,* delims=: " %%a in ("%TIME%") DO SET START=%%c
... some command(s)...
FOR /F "tokens=1,2,* delims=: " %%a in ("%TIME%") DO SET STOP=%%c
Convert null: -format "%[%fx: (%STOP%-START%<0.0)?%STOP%-START%+60.0:%STOP%-START%]" info:
```

This will however only work on Windows versions where the decimal separator is set to the decimal point, as the time is provided according to the locale and `fx` performs its calculations according to the US scheme. In contrast to VBScript, there is no way to change the locale within a batch file (other than changing it for the entire machine via the registry).

As an alternative, one can convert the entire time to hundredths of seconds, which can be done in integer arithmetics, as well as computing any differences later on. The basic code is

```
for /f "tokens=1-4 delims=: , " %%a in ("%TIME%") do ^
set /a Start100S=1%%a * 360000 + 1%%b * 6000 + 1%%c * 100 + 1%%d - 36610100
```

The "1" inserted before each part of the split time code prevents numbers with leading zero from being (mis-)interpreted as octals. The "calculation error" introduced by this way of proceeding is later compensated by subtracting 36610100 in the end. The full (and more elegant) sample code can be found [here](#).

Another approach is the *TimeIt* utility offered by the Windows 2003 Ressource Kit, which can be downloaded [here](#). Officially it however only supports Windows XP.

Guidelines for Batch Programming

These are, in short, the rules to follow when programming batch files:

- Start with `SETLOCAL EnableDelayedExpansion`
- Define a variable to reference IM's Convert tool: `SET IMCONV="%PROGRAMFILES%\ImageMagick"`
- When working with several files, it often simplifies the code when you make the folder that contains the files the current folder, i.e.
 - change to the drive via `%~d1`
 - change to the folder via `CD %~p1`
- Use GnuTools' `iconv.exe` to convert text to UTF-8 and feed text from a file.
- Keep in mind that Convert and Montage write textual output to *stderr*, not *stdout*.
- A lot of errors arise from filenames containing blanks, so check each batch whether it handles such filenames correctly.
- Don't forget to double every percent sign, e.g. when setting the JPEG quality.
- Don't forget to escape special characters in strings, e.g. `"^|"`.
- Integer calculations performed by `SET /A` treat any number starting with zero as octal, thus put a "1" in front of each such number.

Summing it up

The above examples prove that the simple DOS batch file is astonishingly versatile when it is combined with the possibilities offered by ImageMagick. In fact, almost everything can be done in some (crude) way in a batch file. Once you get into thinking in the strange ways followed in the development of the DOS batch file language, scripts can even turn out rather short. Nevertheless, these few line of code will probably have consumed hour of tedious experiments, unless you are really familiar with the batch file language.

If you are aiming at more than basic image processing tasks, it is probably recommendable to use a more sophisticated script language, as the development of the code will turn out to be simpler and more structured.

Visual Basic Script (VBS)

The scripting capabilities of the [Microsoft Windows Script Host](#) (WSH) are more sophisticated than those of the simple batch file language. The WSH is language-independent in the sense that it can make use of different Active Scripting language engines. By default it interprets and runs plain-text JScript files (Java Script) and VBScript files (VisualBasic Script). The Windows Script Host is distributed and installed by default for Windows 98 and newer (it may however have been switched off on a possible target machine because of security concerns). The WSH implements an object model which exposes a set of COM interfaces that allow you to address system objects, especially the file system.

We will not discuss the Windows Script Host in detail over here, as this is done elsewhere (and probably better), but rather give some practical examples how to address typical problems. The examples are given in VisualBasic Script, but the JScript code would be very much alike, thus it should be easy to re-write the examples in JScript, if this is your favourite language.

Like batch files, VBScripts can be written in any editor and I would again suggest Notepad++ as the editor of choice. As for batch files, Microsoft offers no IDE tailored to support the development of VBScripts. There was a Microsoft Script Editor which was shipped with Microsoft Office 2000 through 2003, but I have never tried it. Microsoft also provides the (very rudimentary) Microsoft Script Debugger, but again, I have not much personal experience with it. There are several commercial VBS IDEs offered as shareware at reasonable prices, like [VbsEdit](#).

As has been said in the section [Installing ImageMagick under Windows](#), we will generally not use ImageMagick's COM+ interface in the following. IM's tools such as Convert, Montage and Identify will instead be run directly by invoking the shell's `run` command, with all the needed options and filenames, pretty much the same way it is done in batch files. When assembling the command string, we will however take advantage of the features offered by a real programming language.

A Basic Example: Lens Correction

As the use of the WSH generates some overhead, our start example is not too basic, in order to demonstrate the advantages of VBScript compared to a simple batch file. In the following, we will correct the lens distortion for the Nikon 995 digital camera by use of IM's [barrel distortion](#). The correction parameter(s) depend on the focal length, which is looked up via `identify` first.

For the correction of the Nikon 995 lens, we only need the parameter *b* (i.e. *a*, *c* = 0), which can be calculated from the focal length *f* by:

$$b = 0.000005142 f^3 - 0.000380839 f^2 + 0.009606325 f - 0.075316854$$

This dependency was found by means of the [lensfun database](#) which lists the barrel distortion parameters for this lens. So here is our VBScript:

```
SetLocale(1033)      ' US, i.e. decimal point
const strConv = "IMCONV" ' name of the IM Convert program
const strAdd = "_pt"    ' string attached to the filename
'
Dim wsh
Set wsh = CreateObject("Wscript.Shell")
'
' names of the in- and output files
strFileIn = WScript.Arguments(0)
Pos = InStrRev(strFileIn, ".")
strFileOut = Left(strFileIn, Pos - 1) & strAdd & Mid(strFileIn, Pos)
'
' evaluation of the focal length and calculation of parameter b
command = "identify -format \"%[EXIF:FocalLength]\" \"\" & strFileIn & \"\" \"\"
Set objExec = wsh.Exec(command)
strf = objExec.StdOut.ReadLine
f = eval(strf)
b = 0.000005142 * f * f * f - 0.000380839 * f * f + 0.009606325 * f - 0.075316854
'
Command = strConv & " \"\" & strFileIn _
& \"\" -virtual-pixel black -filter point -distort Barrel \"0.0 \" _
& b & \" 0.0 \" \"\" & strFileOut & \"\" \"\"
wsh.run command, 7, true
```

In the first line we set the locale to US English (1033), which (beside other things) sets the decimal separator to the decimal comma. Otherwise, decimal values would be presented according to the locale, i.e. possibly with a decimal comma instead of a decimal point, which would cause problems when such values are handed over to IM.

The two lines after the definition of the string constants are standard overhead, as we always need a Shell object in order to start IM's programs via its Exec or Run method.

The only script argument is a filename, which we usually provide via Drag & Drop or SendTo. When naming the output file, we append "_pt" to the original filename, just as PTlens would do, e.g. *Photo.JPG* → *Photo_pt.JPG*. The filename is stored in `strFileIn`, from which we derive the name of the output file `strFileOut`. We then run IM's Identify program. The result (i.e. the EXIF rational representing the focal length) is stored in `strf`. EXIF rationals are provided as nominator / denominator, e.g. 82 / 10 = 8.2 mm. The rational thus has to be evaluated before using it in the formula which calculates the parameter *b*.

In the last two lines, we construct the Convert command line and execute the statement via the Run method of the Shell object. The parameter 7 minimises the window and TRUE tells the script to wait for the result.

The above script sketches the general strategy when using VBScript with IM's command line tools (and not the COM+ object). These are called either

- via the Run command of the Shell object, if no textual output is expected
- via the Exec command of the Shell object if textual output has to be evaluated, as typically is the case with Identify.

This first VBScript doesn't do anything that could not have been done by means of a batch file like

```
SETLOCAL EnableDelayedExpansion
FOR /F %i in ('identify -format "%[EXIF:FocalLength]" %1') DO SET FL=%i
FOR /F %i IN ('Convert null: -format "%[fx:0.000005142*(%FL%)^3 - 0.000380839 * (%FL%)^2 + 0.009606325 * %FL% - 0.075316854]" info:') DO SET B=%i
Convert %1 -distort barrel "0.0 !B! 0.0" "%~dpn1_pt~x1"
```

but the VBS code is more straightforward, easier to modify and can easily be extended in regard to error checking and alike. Please note that line breaks in the second FOR statement are impossible, such that we have to leave it as long as it is.

Working with Several Files

One genuine advantage of VBScript in comparison to DOS batch files is that you can easily loop through an arbitrary count of command line arguments. You could for instance pick a number of files in the Windows Explorer and combine the selected images to an index print via IM's Montage. The basic code would be:

```
Dim wsh
Set wsh = CreateObject("Wscript.Shell")
'
strInputFiles = ""
For EACH Arg IN WScript.Arguments
    strInputFiles = strInputFiles & " \"\" & Arg & \"\" \"\"
next
'
IndexFile= Left(WScript.Arguments(1),InStrRev(WScript.Arguments(1),"\") & "index.jpg")
Command = "montage -geometry 210x140+0+5 -tile 6x \" & strInputFiles & \" -quality 80% \"\" & IndexFile & \"\" \"\"
wsh.run command, 7, true
```

The script first concatenates the filenames handed to the script via Drag & Drop. It then derives the name of a JPEG output file in the same folder by some simple string manipulation. At last, Montage is called with appropriate parameters. For larger scripts it is convenient to store the filenames in an array:

```
Dim FName()
Dim wsh
Set wsh = CreateObject("Wscript.Shell")
'
NArgs = WScript.Arguments.Count
Redim FName(NArgs-1)
FOR i = 0 TO NArgs - 1
    FName(i) = \"\" & WScript.Arguments(i) & \"\" \"\"
NEXT
```

First we define a dynamic array via `Dim FName()` and then redimension it via `Redim FName(NArgs-1)`.

The command line of Montage will possibly become very long, because in the input file list, each file is named by its fully qualified filename. The maximum length is **not** the usual 8192 characters, but is determined by the maximum length allowed for a call of the `ShellExecute` Function, which is only 2048 characters for Windows XP. This can cause trouble when the directory name is very long. The error caused **cannot** be handled by the script, as the error occurs **before** the script is executed. A possible solution is to place the files in a local folder with a shorter name. The script-based (partial) solution is the same as for batch files: If only one filename is given, all images in the parent directory are processed. In order to process all GIFs within a folder, we could do something along the lines of:

```
Dim fs, folder
Set fs = CreateObject("Scripting.FileSystemObject")
If WScript.Arguments.Count <> 1 Then WScript.Quit(1)
set Folder = fs.GetFolder(fs.GetParentFolderName(WScript.Arguments(0)))
FN=""
FOR EACH file in folder.files
    If instr(file,"gif") <> 0 THEN FN = FN & File & vbLF
NEXT
MsgBox FN
```

Here we use the `FileSystem` Object to determine the name of the parent folder. The match of the file extension is however checked the ordinary way, as the `Files` object only offers the `Type` property instead.

Quite often, the filenames will have to be sorted alphabetically, as `Drag & Drop` or `SendTo` will pass them to the script in random order. This can be achieved by a bubble sort:

```
for i = 0 to NArgs - 1
    for j = i + 1 to NArgs - 1
        if FName(i) > FName(j) then
            Temp = FName(i)
            FName(i) = FName(j)
            FName(j) = Temp
        end if
    next
next
```

A more sophisticated application of the concept outlined above is presented at the right: A set of video frames has been mounted to two parallel "film strips" by  means of a VBScript and ImageMagick. The perforation gives a visual hint that the progression of the frames is from top to bottom, i.e. column by column, in contrast to the usual western reading pattern left-to-right and then top-to-bottom. The entire script which performs the job can be downloaded from the archive [strip.zip](#).

Marginal note: The red time code at the right top of each frame was generated by an [AVIsynth](#) script. The frames were dumped by exporting them from [VirtualDub](#). With the embedded time code, the frames do not necessarily have to be temporally equidistant, i.e. can be chosen as needed in the Windows Explorer and sent to the script, which is placed in the `SendTo` folder.

Working with Text Files

When working with scripts on a client computer, the input information is generally supplied via `Drag & Drop` or `SendTo`, i.e. basically consists of filenames which will be processed in a manner predefined by the script. Any additional information has either to be supplied by user interaction at runtime or to be supplied in form of a text file.

Basically, we have the following options:

- The script accepts image files as input, accompanied by a (possibly optional) text file, supplying additional information.
- The script accepts a single text file for input, which lists the images to be processed as well as any additional information needed.

In the former case, it is suitable to place the optional text file in the same directory as the images, assigning a standard name to it. The script may then derive the parent directory name from the input images and open the text file in the same directory if present. An example of this approach is the "film strip" mentioned above: At the start of the script we will probably define some standard ordering of the frames, depending of the number of images passed to the script. But there might be scenarios in which we want to deviate from the standard ordering of the frames. Thus we could place a text file named `ordering.txt` in the frames directory, which, if present, will control the ordering of the frames:

```
strTxtFile="ordering.txt"
PDir = fs.GetParentFolderName(FName(0)) & "\"
Wsh.CurrentDirectory = PDir

If fs.FileExists(strTxtFile) then
    Set objFile = fs.OpenTextFile(strTxtFile, 1)
    bCtrlFile = True
    NCols = objFile.ReadLine
    objFile.close
else
    bCtrlFile = False
end if
```

A useful application of the second concept can be found in the perspective mapping of an image to a target plane as [demonstrated](#) on Fred Weinhaus' website.  We could use this concept to "skin" a perspectively distorted image onto a perspectively (mostly) correct version, as demonstrated to the right: In the upper part, the left image shows a photograph taken at the scene of a minor accident. The right photograph was taken at a later visit to the scene, from a somewhat elevated position. In the lower part, the accident photograph (i.e. the planar road surface) is mapped onto the target photograph by means of a perspective transformation. (The aim of this is to visualise the orientation angle of the faint skid mark left by the right front tyre of the black car.)

In order to perform this task, the user has to choose four points in the source image and their target points in the perspectively correct image. We could do this by hand, determining the coordinates in the source image and in the target image by picking the points in an image viewer (like [IrfanView](#)), noting their coordinates and supplying these to an `Convert` command line.

This tedious work can however be simplified by the freeware program [WinMorph](#), which offers a convenient interface to do just this: Pick some source points and their corresponding target points from two pictures. The yellow polylines in the two photographs connect the four chosen points in each picture.

The morphing algorithm itself is however not suited to perform a perspective correction. (The basic functioning of this algorithm is explained in the [Distorts](#) part in the Usage section of the IM website. A demonstration of its usability for morphing is found in Fred Weinhaus' [ShapeMorph](#) script.)

WinMorph stores its information in a structured text file which contains (amongst other information) the filenames of both the source and the target, as well as the coordinates of source and target points. Thus we can derive all information needed for IM's perspective distortion from the WinMorph file.

You can download a copy of all the files and images involved in the file [wmpzr.zip](#).

Piping

So far we have invoked IM's command line tools (like Convert, Identify, Montage) directly via the Run or Exec function of the Shell object. If we however want to use IM's piping capabilities, i.e. feeding one command with the output of the preceeding one, we have to call the command line tools via a command environment. For example, if we want to trim the white borders of the index print generated by the script above, the code would have to be:

```
Dim wsh
Set wsh = CreateObject("Wscript.Shell")
'
strInputFiles = ""
For EACH Arg IN WScript.Arguments
    strInputFiles = strInputFiles & " " & Arg & " "
Next
'
IndexFile= Left(WScript.Arguments(1),InStrRev(WScript.Arguments(1),"\")) & "index.jpg"
Command = "cmd /c montage -geometry 210x140+0+5 -tile 6x " & strInputfiles & " miff:- | convert - -trim "" & IndexFile & """"
wsh.run command, 7, true
```

Here we call the command processor cmd with the option /c in order to close the command box automatically when finished. For debugging purposes, we could also invoke it with the /k option, which leaves the command box open and allows us to read possible error messages.

Testing and Debugging VBScripts

Basically, we are using VBScript to construct the argument list for IM's command line tools, which are then run either themselves or within a DOS box. This means that first of all you should ensure that

- the command line itself does what we expect it to do
- the command line is constructed correctly by the script.

So as a start, you should test the command line itself within a DOS box. When first testing the script, you should not run the IM command, but rather display the text string in a message box via `MsgBox(strCommand)`, because if the command line itself is wrong, there is little any debugging tool could do. The simple message box is also helpful when debugging the script and I never really felt the need for a sophisticated debugger.

Considering runtime testing, you should ensure

- IM's command line tools can be accessed correctly
- the user has selected what you expected her/him to select, i.e. several files (possibly of certain type), a directory, a text file, etc.

Error messages can easily be displayed by the use of `MsgBox(...)`.

Further Information

Unfortunately there is no known tutorial (other than this) which specifically cover using ImageMagick commands in DOS batch files. The web site [DOS "For" Command Help](#) web page has a better explanation of using the "FOR" command.

You may also like to look at [Bonzo's Batch Script](#) page.

Created: 23 April 2009

Updated: 21 March 2012

Author: Wolfgang Hugemann, <ImageMagick_AT_Hugemann.de>

HTML Updated by: [Anthony Thyssen](#), <A.Thyssen_AT_griffith.edu.au>

Examples Generated with: IM v6.7.6-3 2012-03-28

URL: <http://www.imagemagick.org/Usage/windows/>