

# Programming Style & Firmware Standards

Numerous opinions exist in academia and industry regarding programming style rules. One common opinion holds true, through – poorly structured and poorly commented code cannot be maintained, and leads to product quality issues.

In this course, students are expected to submit professional quality code. Students will not be forced to follow a specific code style; however, students who do follow a style guide will likely obtain higher results when their code is graded.

# C Style Guides

- Many C Style Guides exist on the web. Search the web for “C Style Guide”, and you will find a large set of references, including the following:
  - <http://www.chris-lott.org/resources/cstyle/>
  - <http://homepages.inf.ed.ac.uk/dts/pm/Papers/nasa-c-style.pdf>
  - <http://google-styleguide.googlecode.com/svn/trunk/objcguide.xml>
  - [http://www.cs.swarthmore.edu/~newhall/unixhelp/c\\_codestyle.html](http://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html)
  - <http://www.cs.uiowa.edu/~jones/syssoft/style.html>
- A firmware standards manual is available at:
  - <http://www.ganssle.com/>
    - <http://www.ganssle.com/fsm.pdf>
    - <http://www.ganssle.com/commenting.pdf>

# Commenting Style – Header File

Submitted by Andrew Strotheide

```
Dec 21, 07 19:00          5613_io.h          Page 1/2
#ifndef UTIL_5613_IO_H
#define UTIL_5613_IO_H

/*
-----
* Andrew Strotheide
* ECEN 5613, Input/Output Utilities
* Fall 2006, Prof. McClure
* University of Colorado at Boulder
-----
* This file contains the interfaces and definitions for several important
* routines and constants that allow programs to identify and use the input and
* output facilities provided by the underlying hardware on which the library
* is running. Various additional functions are also provided to assist with
* the receipt, handling, and manipulation of input data.
----- */

/* Includes ----- */
#include "5613_io_hardware.h"
#include <stdbool.h>

/* Externalized Variables ----- */
volatile xdata at DEBUG_ADDR unsigned char DEBUG; /* Debug Display */

/* Macros ----- */
#define NEWLINE putchar('\r'); putchar('\n'); /* Print newline combo */

/* Prototypes ----- */

/*
-----
* debug()
*
* Purpose: Writes an 8-bit value to a memory-mapped debug device. This is
* usually a flip-flop, 2-character LED display, or something similar.
* Parms: c - 8-bit character value to be written to debug
* Return: None
----- */
void debug(const unsigned char c);

/*
-----
* putchar()
*
* Purpose: Writes a single character of output to one or more definitions, as
* determined by the presence of one or more PUTCAR_<X> symbols.
* Parms: c - 7-bit character value to be written to output
* Return: None
----- */
void putchar(const char c);

/*
-----
* getchar()
*
* Purpose: Retrieves a single character of input data from the location
* specified by the GETCHAR_<X> symbol. Only one GETCHAR_<X> symbol
* may appear in a compilation unit in order to prevent ambiguity.
* Parms: None
* Return: one character of input data
----- */
char getchar(void);

/*
-----
* safe_gets()
*
* Purpose: This is a version of the popular gets() routine that is safe from
* buffer overruns by limiting the input space. Optionally, it can
* echo received characters back to the sender and it even properly
* handles the backspace key over a terminal emulator. Note that the
* function returns immediately upon filling the buffer or receiving
* and end of line signal. You may wish to ensure the buffer is one
* digit larger than it needs to be and handle that appropriately so
----- */
```

```
Dec 21, 07 19:00          5613_io.h          Page 2/2
* that the Enter key will be required consistently throughout.
* Otherwise, if two characters are possible, on occasions when only
* one is printed, the user will have to finish with Enter, whereas
* when two characters are provided, the function will automatically
* return without pressing Enter. Consistency is good.
* Parms: buffer - location at which buffer is located, upon success, the
* data at this address will be over-written by the response.
* size - maximum number of characters to retrieve (aka, the amount
* of space that was allocated for the buffer).
* echo - Boolean; whether to echo the retrieved data back as output.
* Return: None directly. But upon success, buffer will be filled with the
* retrieved data. Upon failure, buffer will be set to NULL.
----- */
void safe_gets(char* buffer, const unsigned char size, const bool echo);

/*
-----
* get_hex_byte()
*
* Purpose: Prompts the user to provide one or two hexadecimal digits which are
* considered together as one value. The representation is then
* converted from ASCII to hexadecimal and returned in a single byte.
* Valid range is 0-FF and the user may provide one or two digits.
* This function blocks until valid input is provided.
* Parms: prompt - the text displayed to the user to ask for the input
* Return: one byte representing the converted hex value
----- */
unsigned char get_hex_byte(const unsigned char* prompt);

/*
-----
* a_to_uc()
*
* Purpose: Converts the first two of a null-terminated buffer of characters
* into an unsigned char number. This routine is not very robust, but
* it works well within the specified parameters.
* Parms: buffer - location of buffer that holds a the string to convert
* Return: success - numerical value representing the converted string
* failure - 0, which cannot be differentiated from a successful zero
----- */
unsigned char a_to_uc(const unsigned char* buffer);

/*
-----
* aa_to_hex()
*
* Purpose: Converts a two-digit ASCII value into their BCD equivalent byte.
* No validation is performed; results are undefined if either digit
* is anything but [0-9][A-F][a-f].
* Parms: high - high-order digit, formatted as ASCII text
* low - low-order digit, formatted as ASCII text
* Return: numerical value of the hex digits
----- */
unsigned char aa_to_hex(const unsigned char high, const unsigned char low);

/*
-----
* is_all_ascii_hex()
*
* Purpose: Determine whether an entire buffer is comprised of characters whose
* values, when interpreted as ASCII, could all be converted to valid
* hexadecimal digits. Maximum buffer size = 256.
* Parms: buffer - A null-terminated buffer of characters
* Return: true if every character is [0-9][A-F][a-f]; false otherwise
----- */
bool is_all_ascii_hex(const unsigned char* buffer);

/* EOF */
#endif
```

# Commenting Style – Source File

Submitted by Andrew Strotheide

```
Dec 21, 07 19:00      5613_io.c      Page 1/4
/* -----
 * Andrew Strotheide
 * ECEN 5613, Input/Output Utilities
 * Fall 2006, Prof. McClure
 * University of Colorado at Boulder
 * -----
 * This file contains the implementations for several important routines that
 * allow programs to identify and use the input and output facilities provided
 * by the underlying hardware on which the library is running, as well as a
 * number of convenient functions for input manipulation. For more
 * information, please see the master file, 5613_io.h
 * ----- */

/* Includes ----- */
#include "5613_io.h"
/* Only include serial header when necessary */
#if (defined GETCHAR_SERIAL || defined PUTCHAR_SERIAL)
#include "5613_seriah"
#endif
#include <stdio.h>
#include <stdbool.h>

/* ----- */
void debug(const unsigned char c)
{
#ifdef NO_DEBUG
    DEBUG = c;
#endif
}

/* ----- */
void putchar(const char c)
{
    c; /* Prevent compiler from optimizing this away */
    /* Serial port output */
#ifdef PUTCHAR_SERIAL
    util_serial_putc(c); /* Andrew's non-interrupt-based serial output */
    //serial_putc(c); /* SDCC built-in interrupt-based serial output */
#endif
    /* Debug display output */
#ifdef PUTCHAR_DEBUG
    DEBUG = c;
#endif
    /* LCD display output */
#ifdef PUTCHAR_LCD
#warning putchar() cannot write to LCD; unimplemented.
#endif
    #if (!defined PUTCHAR_LCD && !defined PUTCHAR_DEBUG && !defined PUTCHAR_SERIAL)
#warning putchar() is null; no output device specified
    #endif
}

/* ----- */
#ifdef GETCHAR_SERIAL
char getchar(void)
{
    char c = 0;
    /* Serial port input */
    {
        c = util_serial_getc(); /* Non-interrupt-based serial input */
        //c = serial_getc(); /* SDCC built-in interrupt-based serial input */
    }
    return (c);
}
#else
#warning getchar() unavailable; no input method specified.
#endif
/* ----- */
```

```
Dec 21, 07 19:00      5613_io.c      Page 2/4
void safe_gets(char* buffer, const unsigned char size, const bool echo)
{
    bool finished = false;
    unsigned char i = 0;
    unsigned char temp;

    /* Size must be > 1 to allow for null char and some input. */
    if (size > 1)
    {
        /* Get input until max size reached or newline encountered. */
        while ( ( i < (size - 1) ) && (!finished) )
        {
            temp = getchar();
            if (temp == '\r' || temp == '\n')
            {
                /* Newline encountered; set flag to bail out. */
                finished = true;
            }
            /* Process backspace or delete if other chars in buffer. */
            else if (temp == 127 || temp == 8)
            {
                /* Only allow deletion up to first character, no more. */
                if (i > 0)
                {
                    /* Echo the key's effects, if necessary. */
                    if (echo)
                    {
                        putchar(temp); /* Rewind the display cursor. */
                        putchar(' '); /* Blank out the deleted character. */
                        putchar(temp); /* Rewind the display cursor again. */
                    }
                    i--; /* Backup i to update remove last char from buffer */
                }
            }
            else
            {
                /* Add character to buffer, then increment to next position. */
                buffer[i++] = temp;
                if (echo)
                {
                    putchar(temp); /* Echo the value if necessary. */
                }
            }
        }
        buffer[i] = '\0'; /* Null-terminate the obtained string */
    }
    else
    {
        /* There was a problem; bail out with empty string. */
        buffer = '\0';
    }
}

/* ----- */
unsigned char get_hex_byte(const unsigned char* prompt)
{
    unsigned char buffer[3];

    /* Get selections from user; use a buffer-overload-immune gets() function
     * that guarantees null-terminated results. Keep asking for input until
     * the user gets it right. */
    while (true)
    {
        /* Prompt for data */
        puts(prompt);
        putchar('\r');
        putchar('>');
        putchar(' ');
        buffer[0] = '\0';
    }
}

/* ----- */
```

# Commenting Style – Table of Contents

Submitted by Andrew Strotheide

Dec 22, 07 15:02	Table of Content	Page 1/1
<b>Table of Contents</b>		
1	Makefile..... sheets	0 to 0 ( 1) pages 0- 0 1 lines
2	5613_io_hardware.h... sheets	1 to 1 ( 1) pages 1- 2 72 lines
3	5613_io.h..... sheets	2 to 2 ( 1) pages 3- 4 135 lines
4	5613_io.c..... sheets	3 to 4 ( 2) pages 5- 8 216 lines
5	5613_serial.h..... sheets	5 to 5 ( 1) pages 9- 9 67 lines
6	5613_serial.c..... sheets	6 to 6 ( 1) pages 10- 11 81 lines
7	5613_delay.h..... sheets	7 to 7 ( 1) pages 12- 12 54 lines
8	5613_delay.c..... sheets	8 to 9 ( 2) pages 13- 15 182 lines
9	5613_print.h..... sheets	10 to 10 ( 1) pages 16- 16 32 lines
10	5613_print.c..... sheets	11 to 11 ( 1) pages 17- 17 47 lines
11	lcd_hardware.h..... sheets	12 to 12 ( 1) pages 18- 19 94 lines
12	lcd_commands.h..... sheets	13 to 13 ( 1) pages 20- 21 77 lines
13	lcd_driver.h..... sheets	14 to 15 ( 2) pages 22- 24 177 lines
14	lcd_driver.c..... sheets	16 to 18 ( 3) pages 25- 30 376 lines
15	lcd_driver_test.c..... sheets	19 to 20 ( 2) pages 31- 33 156 lines
16	i2c_hardware.h..... sheets	21 to 21 ( 1) pages 34- 35 74 lines
17	i2c_driver.h..... sheets	22 to 23 ( 2) pages 36- 39 219 lines
18	i2c_driver.c..... sheets	24 to 27 ( 4) pages 40- 47 514 lines
19	i2c_eeprom.h..... sheets	28 to 29 ( 2) pages 48- 50 202 lines
20	i2c_eeprom.c..... sheets	30 to 31 ( 2) pages 51- 53 157 lines
21	i2c_expander.h..... sheets	32 to 32 ( 1) pages 54- 55 90 lines
22	i2c_expander.c..... sheets	33 to 33 ( 1) pages 56- 57 82 lines
23	i2c_rtc.h..... sheets	34 to 36 ( 3) pages 58- 63 394 lines
24	i2c_rtc.c..... sheets	37 to 38 ( 2) pages 64- 66 181 lines
25	clock.h..... sheets	39 to 40 ( 2) pages 67- 69 160 lines
26	clock.c..... sheets	41 to 43 ( 3) pages 70- 74 297 lines
27	test_simple_timer.c..... sheets	44 to 44 ( 1) pages 75- 75 36 lines
28	test_eeprom.c..... sheets	45 to 46 ( 2) pages 76- 78 189 lines
29	test_expander.c..... sheets	47 to 48 ( 2) pages 79- 82 246 lines
30	test_rtc.c..... sheets	49 to 50 ( 2) pages 83- 85 149 lines
31	main_demo.c..... sheets	51 to 54 ( 4) pages 86- 93 526 lines

Many thanks to former student, Andrew Strotheide, who provided the “pretty\_pdf.bat” tool, described on the following pages. That tool was used to create the formatted output shown on the previous pages.

From: Andrew Strotheide  
To: Prof. Linden McClure  
Subject: Source code formatting using a2ps under Windows  
Date: Monday, January 07, 2008 1:43 PM

Prof. McClure,

Here's the information about the source code formatter.

When I did my undergraduate work at CU, the Computer Science department required us to print out our source code using "a2ps", a common UNIX tool which converts "anything" to PostScript. a2ps handles all of the source code formatting as well as the table of contents and it is very easy to use. The undergrad lab had only UNIX machines and PostScript printers, so it was a very simple matter to run the program and direct the output directly to the printer. I continue to use the program today when I need "pretty-printed" source code, as it is available by default on Mac OSX and Linux. It can also be used from Windows, although it requires a bit more work, which is the nature of this email.

It occurred to me that it would probably be incredibly helpful to you if all of your future students were to use the same program for formatting and printing their source code output. I've put together some information that will allow you to install the required programs on the Windows lab machines and use them to generate PDF files of the source code that can be easily viewed and printed using Adobe Reader. My primary goals here were to make it as easy as possible to install and run the software, despite the quirk of using it under Windows instead of UNIX. If you decide to use this and you or your lab techs need any further information, please let me know.

Basically, here's what's involved. You have to download and install a2ps and GhostScript. Although these programs can be used directly, it's tedious. Therefore, I created the attached batch file program to wrap the ugliness of the underlying interfaces so that you can create PDF files in the same format as the one I gave you, using a very simple interface. All of the required programs are free. Mine has a BSD license and the others have GPL licenses. The attached file is .txt; you need to save it as a .bat file instead (I had to rename it to get past the University's email scanner).

Installation:

1) Download and install a2ps

<http://gnuwin32.sourceforge.net/downloads/a2ps.php>

2) Download and install GhostScript

<http://mirror.cs.wisc.edu/pub/mirrors/ghost/GPL/g860/g860w32.exe>

3) Save the attached .txt file as **pretty\_pdf.bat** file on the target machine. Place it either in the a2ps binary directory, or another directory of your choosing.

4) Add the bin directories for a2ps and GhostScript, and the location of the batch file to the system path so that they will be accessible to all users from the command line.

Usage:

From a command prompt, execute the batch program as follows:

**pretty\_pdf <output\_file.pdf> input\_file1 input\_file2 ...**

Wildcards (such as \*.h \*.c) are valid here.

The result is that all of the input files will be processed by a2ps and combined (in the order provided) into a nicely formatted PostScript file with a table of contents. Then the PostScript will be converted automatically to PDF using GhostScript. At that point, the file can be emailed for electronic submission, viewed using Adobe Reader, or printed to any Windows printer, regardless of whether it speaks PostScript, PCL, etc.

It is certainly possible to call a2ps directly instead of using my wrapper program. However, a2ps cannot handle any wildcards, so if using a2ps directly, all files must be listed with their complete names. Consult the batch file program for the specific usage requirements of a2ps and GhostScript. If you use GhostScript to convert to PDF as I've provided with my batch file, you won't need a PostScript viewer. However, if you want one anyway, you can get GSVIEW here:

**<http://mirror.cs.wisc.edu/pub/mirrors/ghost/ghostgum/gsv49w32.exe>**

As another aside, I should mention that you could also set this up by downloading the cygwin environment and during install, indicating a desire to add the a2ps and GhostScript packages. Then you can just use everything straight from cygwin. However, using cygwin adds another level of complexity that I don't think you'll want to deal with, which is why I didn't describe that process.

This may be far more information than you want or need, but given how potentially useful this could be for your classes, I didn't want to leave you hanging by simply saying, "I used a2ps; go get it and figure it out." This should easily allow you to obtain, install, and use it. I hope it helps.

Andrew Strotheide

```

@echo off
:: -----
:: pretty_pdf.bat
:: -----
:: Copyright (c) 2008, Andrew Strotheide (andrew@strotheide.com)
:: All rights reserved.
::
:: Redistribution and use in source and binary forms, with or without
:: modification, are permitted provided that the following conditions are met:
::
:: * Redistributions of source code must retain the above copyright notice
:: this list of conditions and the following disclaimer.
:: * Redistributions in binary form must reproduce the above copyright
:: notice, this list of conditions and the following disclaimer in the
:: documentation and/or other materials provided with the distribution.
:: * Neither the name of Andrew Strotheide nor the names of other
:: contributors may be used to endorse or promote products derived from
:: this software without specific prior written permission.
::
:: THIS SOFTWARE IS PROVIDED BY ANDREW STROTHEIDE "AS IS" AND ANY EXPRESS OR
:: IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
:: MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
:: EVENT SHALL ANDREW STROTHEIDE BE LIABLE FOR ANY DIRECT, INDIRECT,
:: INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
:: LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
:: OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
:: LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
:: NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
:: EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
:: -----
::
:: Description:
:: -----
:: This program is a wrapper to pretty-print files using a2ps and convert the
:: formatted PostScript output to PDF using GhostScript (on Windows, without
:: Cygwin). The most important feature is the support for wildcards in the
:: list of files to be pretty-printed. This is normally difficult because
:: a2ps was originally written for UNIX and expects wildcards to have already
:: been expanded by the command interpreter before program invocation, whereas
:: DOS and Windows programs receive the wildcard characters directly and must
:: perform their own expansions. Since a2ps cannot handle wildcards that have
:: not been expanded, this batch program provides a rudimentary means by which
:: to expand the names before providing them to a2ps. Also, the output from
:: a2ps is a PostScript file, which is generally of little use when running
:: under Windows. Therefore, this program takes the next step of converting
:: the output from PostScript to PDF so it is accessible from programs such
:: as Adobe Reader.
::
:: Prerequisites:
:: -----
:: In order to use this program, you must install some third-party software.
:: Installation of such software is subject to the license agreements of each
:: package, which is almost certainly different than the license under which
:: this batch program is provided. The following programs are required:
:: * a2ps, which is available as part of the gnuwin32 suite of programs on
:: SourceForge, but can be installed standalone
:: (http://gnuwin32.sourceforge.net/packages/a2ps.htm) .

```

```

:: * GhostScript, which has two distributions: AFPL or GNU license
:: (http://pages.cs.wisc.edu/~ghost/).
:: * Optionally, you may wish to install a PostScript viewer, such as GSView
:: which will allow you to preview pure PostScript files and convert or
:: print them manually. GSView requires GhostScript and is available from
:: from the GhostScript link above.
::
:: Usage:
:: -----
:: pretty_pdf <output_file.pdf> input_files ...
:: * The output file must have a .pdf extension and may optionally include
:: a full pathname.
:: * One or more input file must be provided.
:: * Wildcard characters are acceptable in the input file listing (ex. *.c)
:: * Executables for GhostScript and a2ps must be accessible directly from
:: the command line, so their binary folders must be in the PATH variable
:: (see http://www.computerhope.com/issues/ch000549.htm for assistance).
::
:: -----
:: Initialization
:: -----
SET EXITCODE=0
SETLOCAL ENABLEDELAYEDEXPANSION
SET USAGE=Usage: %0 {output_file.pdf} input_file1 [input_file2 ...]
:: -----
:: Errorlevels
:: -----
SET EXIT_NO_OUTPUT_FILE=1
SET EXIT_OUTPUT_FILE_EXT=2
SET EXIT_NO_OUTPUT_PATH=3
SET EXIT_NO_INPUT_FILES=4
SET EXIT_BAD_INPUT_FILES=5
SET EXIT_PS_EXISTS=6
SET EXIT_GENERATE=7
SET EXIT_PS_MISSING=8
SET EXIT_PDF_EXISTS=9
SET EXIT_CONVERT=10
SET EXIT_PDF_MISSING=11
:: -----
:: Validate output file
:: -----
IF ()==(~1) GOTO Error_No_Output_File
IF NOT EXIST (%~dp1) GOTO Error_No_Output_Path 2> NUL
IF NOT (.pdf)==(~x1) GOTO Error_Output_File_Ext
SET PS_NAME=%~f1.ps"
SET PDF_NAME=%~f1"
IF EXIST %PS_NAME% GOTO Error_PS_Exists
IF EXIST %PDF_NAME% GOTO Error_PDF_Exists
SHIFT
:: -----
:: Validate input files
:: -----

```

```

IF ()=(%) GOTO Error_No_Input_Files
SET INPUT_FILES=
FOR /F %%F in ('dir /l/b/-p %*') DO SET INPUT_FILES=!INPUT_FILES! "%%F"
IF NOT ERRORLEVEL 0 GOTO Error_Bad_Input_Files

:: -----
:: Use a2ps to create PostScript code listing
:: -----
:Generate
ECHO Generating temporary PostScript file...
a2ps %INPUT_FILES% --medium=letter --toc -o %PS_NAME%
IF NOT ERRORLEVEL 0 GOTO Error_Generate
IF NOT EXIST %PS_NAME% GOTO Error_PS_Missing

:: -----
:: Use GhostScript to convert PostScript to PDF
:: -----
:Convert
ECHO Generating PDF file %PDF_NAME%...
gswin32c -dNOPAUSE -sDEVICE=pdfwrite -sOutputFile=%PDF_NAME% -q %PS_NAME% -c quit
IF NOT ERRORLEVEL 0 GOTO Error_Convert
IF NOT EXIST %PDF_NAME% GOTO Error_PDF_Missing
IF EXIST %PS_NAME% DEL %PS_NAME%
GOTO End

:: -----
:: Error Conditions
:: -----
:Error_No_Output_File
ECHO *** Error: No output file specified.
ECHO %USAGE%
SET EXITCODE=%EXIT_NO_OUTPUT_FILE%
GOTO End

:Error_Output_File_Ext
ECHO *** Error: Output file extension must be ".pdf".
ECHO %USAGE%
SET EXITCODE=%EXIT_OUTPUT_FILE_EXT%
GOTO End

:Error_No_Output_Path
ECHO *** Error: Output file path "%~dp1" does not exist.
ECHO %USAGE%
SET EXITCODE=%EXIT_NO_OUTPUT_PATH%
GOTO End

:Error_No_Input_Files
ECHO *** Error: No input files provided.
ECHO %USAGE%
SET EXITCODE=%EXIT_NO_INPUT_FILES%
GOTO End

:Error_Bad_Input_Files
ECHO *** Error: Input file list contains invalid item(s).
ECHO %USAGE%
SET EXITCODE=%EXIT_BAD_INPUT_FILES%
GOTO End

```

```

:Error_PS_Exists
ECHO *** Error: Cannot overwrite PostScript file %PS_NAME%; remove it or use a
different output name.
ECHO %USAGE%
SET EXITCODE=%EXIT_PS_EXISTS%
GOTO End

:Error_Generate
ECHO *** Error: A2PS returned an error.
SET EXITCODE=%EXIT_GENERATE%
GOTO End

:Error_PS_Missing
ECHO *** Error: The PostScript file was not generated.
SET EXITCODE=%EXIT_PS_MISSING%
GOTO End

:Error_PDF_Exists
ECHO *** Error: Cannot overwrite PDF file %PDF_NAME%; remove it or use a different
output name.
SET EXITCODE=%EXIT_PDF_EXISTS%
GOTO End

:Error_Convert
ECHO *** Error: GhostScript returned an error.
SET EXITCODE=%EXIT_CONVERT%
GOTO End

:Error_PDF_Missing
ECHO *** Error: The PDF file was not created.
SET EXITCODE=%EXIT_PDF_MISSING%
GOTO End

:: -----
:: Clean up and exit
:: -----
:End
EXIT /B %EXITCODE%

```