CUPS Configuration Management Plan

CUPS−CMP−1.1

Easy Software Products

Table of Contents

Table of Contents

1 Scope

1.1 Identification

This configuration management plan document provides the guidelines for development and maintenance of the Common UNIX Printing System ("CUPS") Version 1.1 software.

1.2 System Overview

CUPS provides a portable printing layer for UNIX®−based operating systems. It has been developed by Easy Software Products to promote a standard printing solution for all UNIX vendors and users. CUPS provides the System V and Berkeley command−line interfaces.

CUPS uses the Internet Printing Protocol ("IPP") as the basis for managing print jobs and queues. The Line Printer Daemon ("LPD") Server Message Block ("SMB"), and AppSocket (a.k.a. JetDirect) protocols are also supported with reduced functionality. CUPS adds network printer browsing and PostScript Printer Description ("PPD") based printing options to support real−world printing under UNIX.

CUPS also includes a customized version of GNU Ghostscript (currently based off GNU Ghostscript 5.50) and an image file RIP that are used to support non−PostScript printers. Sample drivers for HP and EPSON printers are included that use these filters.

1.3 Document Overview

This configuration management document is organized into the following sections:

- 1 − Scope
- 2 − References
- 3 − File Management
- 4 − Trouble Report Processing
- 5 − Software Releases
- A − Glossary
- B − Coding Requirements

2 References

2.1 CUPS Documentation

The following CUPS documentation is referenced by this document:

- CUPS−CMP−1.1: CUPS Configuration Management Plan
- CUPS−IDD−1.1: CUPS System Interface Design Description
- CUPS−IPP−1.1: CUPS Implementation of IPP
- CUPS−SAM−1.1.x: CUPS Software Administrators Manual
- CUPS−SDD−1.1: CUPS Software Design Description
- CUPS−SPM−1.1.x: CUPS Software Programming Manual
- CUPS−SSR−1.1: CUPS Software Security Report
- CUPS−STP−1.1: CUPS Software Test Plan
- CUPS−SUM−1.1.x: CUPS Software Users Manual
- CUPS−SVD−1.1: CUPS Software Version Description

2.2 Other Documents

The following non−CUPS documents are referenced by this document:

- Adobe PostScript Printer Description File Format Specification, Version 4.3.
- Adobe PostScript Language Reference, Third Edition.
- IPP: Job and Printer Set Operations
- IPP/1.1: Encoding and Transport
- IPP/1.1: Implementers Guide
- IPP/1.1: Model and Semantics
- RFC 1179, Line Printer Daemon Protocol
- RFC 2567, Design Goals for an Internet Printing Protocol
- RFC 2568, Rationale for the Structure of the Model and Protocol for the Internet Printing Protocol
- RFC 2569, Mapping between LPD and IPP Protocols
- RFC 2616, Hypertext Transfer Protocol −− HTTP/1.1
- RFC 2617, HTTP Authentication: Basic and Digest Access Authentication

3 File Management

3.1 Directory Structure

Each source file shall be placed a sub−directory corresponding to the software sub−system it belongs to ("scheduler", "cups", etc.) To remain compatible with older UNIX filesystems, directory names shall not exceed 16 characters in length.

3.2 Source Files

Source files shall be documented and formatted as described in Appendix B, Coding Requirements.

3.3 Configuration Management

Source files shall be placed under the control of the Concurrent Versions System ("CVS") software. Source files shall be "checked in" with each change so that modifications can be tracked.

Documentation on the CVS software is included with the whitepaper, "CVS II: Parallelizing Software Development".

4 Trouble Report Processing

A Software Trouble Report ("STR") shall be submitted every time a user or vendor experiences a problem with the CUPS software. Trouble reports are maintained in a database with one of the following states:

1. STR is closed with complete resolution
2. STR is closed without resolution
3. STR is active
4. STR is pending (new STR or additional information available)

Trouble reports shall be processed using the following steps.

4.1 Classification

When a trouble report is received it must be classified at one of the following levels:

1. Request for enhancement
2. Documentation error
3. Unable to print a file
4. Unable to print to a printer
5. Unable to print at all

The scope of the problem should also be determined as:

1. Specific to a machine
2. Specific to an operating system
3. Applies to all machines and operating systems

4.2 Identification

Once the level and scope of the trouble report is determined the software sub−system(s) involved with the problem are determined. This may involve additional communication with the user or vendor to isolate the problem to a specific cause.

When the sub−system(s) involved have been identified, an engineer will then determine the change(s) needed and estimate the time required for the change(s).

4.3 Correction

Corrections are scheduled based upon the severity and complexity of the problem. Once all changes have been made, documented, and tested successfully a new software release snapshot is generated. Additional tests are added as necessary for proper testing of the changes.

4.4 Notification

The user or vendor is notified when the fix is available or if the problem was caused by user error.

5 Software Releases

5.1 Version Numbering

CUPS uses a three−part version number separated by periods to represent the major, minor, and patch release numbers:

```
major.minor.patch
1.1.0
```

Beta−test releases are indentified by appending the letter B followed by the build number:

```
major.minor.patchbbuild
1.1.0b1
```

A CVS snapshot is generated for every beta and final release and uses the version number preceded by the letter "v" and with the decimal points replaced by underscores:

```
v1_0_0b1
v1_0_0
```

Each change that corrects a fault in a software sub−system increments the patch release number. If a change affects the software design of CUPS then the minor release number will be incremented and the patch release number reset to 0. If CUPS is completely redesigned the major release number will be incremented and the minor and patch release numbers reset to 0:

```
1.1.0b1    First beta release
1.1.0b2    Second beta release
1.1.0      First production release
1.1.1b1    First beta of 1.1.1
1.1.1      Production release of 1.1.1
1.1.1b1    First beta of 1.1.1
1.1.1      Production release of 1.1.1
2.0.0b1    First beta of 2.0.0
2.0.0      Production release of 2.0.0
```

5.2 Generation

Software releases shall be generated for each successfully completed software trouble report. All object and executable files shall be deleted prior to performing a full build to ensure that source files are recompiled.

5.3 Testing

Software testing shall be conducted according to the CUPS Software Test Plan, CUPS−STP−1.1. Failed tests cause STRs to be generated to correct the problems found.

5.4 Release

When testing has been completed successfully a new distribution image is created from the current CVS code "snapshot". No production release shall contain software that has not passed the appropriate software tests.

A Glossary

A.1 Terms

C
: A computer language.

parallel
: Sending or receiving data more than 1 bit at a time.

pipe
: A one−way communications channel between two programs.

serial
: Sending or receiving data 1 bit at a time.

socket
: A two−way network communications channel.

A.2 Acronyms

ASCII
: American Standard Code for Information Interchange

CUPS
: Common UNIX Printing System

ESC/P
: EPSON Standard Code for Printers

FTP
: File Transfer Protocol

HP−GL
: Hewlett−Packard Graphics Language

HP−PCL
: Hewlett−Packard Page Control Language

HP−PJL
: Hewlett−Packard Printer Job Language

IETF
: Internet Engineering Task Force

IPP
: Internet Printing Protocol

ISO
: International Standards Organization

LPD
: Line Printer Daemon

MIME
: Multimedia Internet Mail Exchange

PPD
: PostScript Printer Description

SMB
: Server Message Block

TFTP
: Trivial File Transfer Protocol

B Coding Requirements

These coding requirements provide detailed information on source file formatting and documentation content. These guidelines shall be applied to all C and C++ source files provided with CUPS.

B.1 Source Files

B.1.1 Naming

All source files names shall be 16 characters or less in length to ensure compatibility with older UNIX filesystems. Source files containing functions shall have an extension of ".c" for ANSI C and ".cxx" for C++ source files. All other "include" files shall have an extension of ".h".

B.1.2 Documentation

The top of each source file shall contain a header giving the name of the file, the purpose or nature of the source file, the copyright and licensing notice, and the functions contained in the file. The file name and revision information is provided by the CVS "Id" tag:

```
/*
 * "$Id$"
 *
 *   Description of file contents.
 *
 *   Copyright 1997−2001 by Easy Software Products, all rights
 *   reserved.
 *
 *   These coded instructions, statements, and computer programs are
 *   the property of Easy Software Products and are protected by
 *   Federal copyright law.  Distribution and use rights are outlined
 *   in the file "LICENSE.txt" which should have been included with
 *   this file.  If this file is missing or damaged please contact
 *   Easy Software Products at:
 *
 *       Attn: CUPS Licensing Information
 *       Easy Software Products
 *       44141 Airport View Drive, Suite 204
 *       Hollywood, Maryland 20636−3111 USA
 *
 *       Voice: (301) 373−9600
 *       EMail: cups−info@cups.org
 *         WWW: http://www.cups.org
 *
 * Contents:
 *
 *   function1() − Description 1.
 *   function2() − Description 2.
 *   function3() − Description 3.
 */
```

The bottom of each source file shall contain a trailer giving the name of the file using the CVS "Id" tag. The primary purpose of this is to mark the end of a source file; if the trailer is missing it is possible that code has been lost near the end of the file:

```
/*
 * End of "$Id$".
 */
```

B.2 Functions

B.2.1 Naming

Functions with a global scope shall be capitalized ("DoThis", "DoThat", "DoSomethingElse", etc.) The only exception to this rule shall be the CUPS interface library functions which may begin with a prefix word in lowercase ("cupsDoThis", "cupsDoThat", etc.)

Functions with a local scope shall be declared "static" and be lowercase with underscores between words ("do_this", "do_that", "do_something_else", etc.)

B.2.2 Documentation

Each function shall begin with a comment header describing what the function does, the possible input limits (if any), and the possible output values (if any), and any special information needed:

```
/*
 * 'do_this()' - Compute y = this(x).
 *
 * Notes: none.
 */

static float     /* O - Inverse power value, 0.0 <= y <= 1.1 */
do_this(float x) /* I - Power value (0.0 <= x <= 1.1) */
{
  ...
  return (y);
}
```

B.3 Methods

B.3.1 Naming

Methods shall be in lowercase with underscores between words ("do_this", "do_that", "do_something_else", etc.)

B.3.2 Documentation

Each method shall begin with a comment header describing what the method does, the possible input limits (if any), and the possible output values (if any), and any special information needed:

```
/*
 * 'class::do_this()' - Compute y = this(x).
 *
 * Notes: none.
 */

float                   /* O - Inverse power value, 0.0 <= y <= 1.0 */
class::do_this(float x) /* I - Power value (0.0 <= x <= 1.0) */
```

```
{
  ...
  return (y);
}
```

B.4 Variables

B.4.1 Naming

Variables with a global scope shall be capitalized ("ThisVariable", "ThatVariable", "ThisStateVariable", etc.) The only exception to this rule shall be the CUPS interface library global variables which must begin with the prefix "cups" ("cupsThisVariable", "cupsThatVariable", etc.) Global variables shall be replaced by function arguments whenever possible.

Variables with a local scope shall be lowercase with underscores between words ("this_variable", "that_variable", etc.) Any local variables shared by functions within a source file shall be declared "static".

B.4.2 Documentation

Each variable shall be declared on a separate line and shall be immediately followed by a comment block describing the variable:

```
int this_variable;   /* The current state of this */
int that_variable;   /* The current state of that */
```

B.5 Types

B.5.1 Naming

All type names shall be lowercase with underscores between words and "_t" appended to the end of the name ("this_type_t", "that_type_t", etc.)

B.5.2 Documentation

Each type shall have a comment block immediately before the typedef:

```
/*
 * This type is for CUPS foobar options.
 */
typedef int cups_this_type_t;
```

B.6 Structures

B.6.1 Naming

All structure names shall be lowercase with underscores between words and "_str" appended to the end of the name ("this_struct_str", "that_struct_str", etc.)

B.6.2 Documentation

Each structure shall have a comment block immediately before the struct and each member shall be documented in accordance with the variable naming policy above:

```
/*
 * This structure is for CUPS foobar options.
 */
struct cups_this_struct_str
{
  int this_member;   /* Current state for this */
  int that_member;   /* Current state for that */
};
```

B.7 Classes

B.7.1 Naming

All class names shall be lowercase with underscores between words ("this_class", "that_class", etc.)

B.7.2 Documentation

Each class shall have a comment block immediately before the class and each member shall be documented in accordance with the variable naming policy above:

```
/*
 * This class is for CUPS foobar options.
 */
class cups_this_class
{
  int this_member;   /* Current state for this */
  int that_member;   /* Current state for that */
};
```

B.8 Constants

B.8.1 Naming

All constant names shall be uppercase with underscored between words ("THIS_CONSTANT", "THAT_CONSTANT", etc.) Constants defined for the CUPS interface library must begin with an uppercase prefix ("CUPS_THIS_CONSTANT", "CUPS_THAT_CONSTANT", etc.)

Typed enumerations shall be used whenever possible to allow for type checking by the compiler.

B.8.2 Documentation

Comment blocks shall immediately follow each constant:

```
enum
{
  CUPS_THIS_TRAY,   /* This tray */
  CUPS_THAT_TRAY    /* That tray */
};
```

B.9 Code

B.9.1 Documentation

All source code shall utilize block comments within functions to describe the operations being performed by a group of statements:

```
/*
 * Clear the state array before we begin...
 */

for (i = 0; i < (sizeof(array) / sizeof(sizeof(array[0])); i ++)
  array[i] = STATE_IDLE;

/*
 * Wait for state changes...
 */

do
{
  for (i = 0; i < (sizeof(array) / sizeof(sizeof(array[0])); i ++)
    if (array[i] != STATE_IDLE)
      break;

  if (i == (sizeof(array) / sizeof(array[0])))
    sleep(1);
} while (i == (sizeof(array) / sizeof(array[0])));
```

B.9.2 Style

B.9.2.a Indentation

All code blocks enclosed by brackets shall begin with the opening brace on a new line. The code then follows starting on a new line after the brace and is indented 2 spaces. The closing brace is then placed on a new line following the code at the original indentation:

```
{
  int i; /* Looping var */

 /*
  * Process foobar values from 0 to 999...
  */

  for (i = 0; i < 1000; i ++)
  {
    do_this(i);
    do_that(i);
  }
}
```

Single−line statements following "do", "else", "for", "if", and "while" shall be indented 2 spaces as well. Blocks of code in a "switch" block shall be indented 4 spaces after each "case" and "default" case:

```
switch (array[i])
```

```
{
  case STATE_IDLE :
      do_this(i);
      do_that(i);
      break;
  default :
      do_nothing(i);
      break;
}
```

B.9.2.b Spacing

A space shall follow each reserved word ("if", "while", etc.) Spaces shall not be inserted between a function name and the arguments in parenthesis.

B.9.2.c Return Values

Parenthesis shall surround values returned from a function using "return":

```
return (STATE_IDLE);
```

B.9.2.d Loops

Whenever convenient loops should count downward to zero to improve program performance:

```
for (i = sizeof(array) / sizeof(array[0]) - 1; i >= 0; i --)
  array[i] = STATE_IDLE;
```

C Software Trouble Report Form

Summary of Problem:	______________________________

Problem Severity:	__1=RFE
	__2=Documentation−Error
	__3=Unable−to−Print−a−File
	__4=Unable−to−Print−to−a−Printer
	__5=Unable−to−Print−at−All

Problem Scope:	__1=Machine __2=Operating−System __3=All
Detailed Description of Problem:	______________________________ ______________________________ ______________________________ ______________________________ ______________________________ ______________________________