[1] [] 1 [Generated on Tue Jan 8 15:33:02 2013 for APDK Developer's Guide by Doxygen] []Generated on Tue Jan 8 15:33:02 2013 for APDK Developer's Guide by Doxygen

[1] [] 1 [Generated on Tue Jan 8 15:33:02 2013 for APDK Developer's Guide by Doxygen] []Generated on Tue Jan 8 15:33:02 2013 for APDK Developer's Guide by Doxygen

APDK Developer's Guide Reference Manual

04.11.00

Generated by Doxygen 1.2.15

Tue Jan 8 15:33:00 2013

Contents

1 APDK Developer's Guide Module Index

1.1 APDK Developer's Guide Modules

Here is a list of all modules:

2 APDK Developer's Guide Namespace Index

2.1 APDK Developer's Guide Namespace List

Here is a list of all documented namespaces with brief descriptions:

3 APDK Developer's Guide Hierarchical Index

3.1 APDK Developer's Guide Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 APDK Developer's Guide Compound Index

4.1 APDK Developer's Guide Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 APDK Developer's Guide Page Index

5.1 APDK Developer's Guide Related Pages

Here is a list of all related documentation pages:

6 APDK Developer's Guide Module Documentation

6.1 Globals

Enumerations

- enum **PEN_TYPE** { **BLACK_PEN**, **COLOR_PEN**, **BOTH_PENS**, **MDL_PEN**, **MDL_BOTH**, **MDL_AND_BLACK_PENS**, **MDL_BLACK_AND_COLOR_PENS**, **GREY_PEN**, **GREY_BOTH**, **MDL_AND_GREY_PENS**, **MDL_GREY_AND_COLOR_PENS**, **UNKNOWN_PEN**, **NO_PEN**, **DUMMY_PEN**, **MAX_PEN_TYPE** = UNKNOWN_PEN }

  Possible pen combinations.

- enum **PAPER_SIZE** { **UNSUPPORTED_SIZE** = -1, **LETTER** = 0, **A4** = 1, **LEGAL** = 2, **PHOTO_SIZE** = 3, **A6** = 4, **CARD_4x6** = 5,

B4 = 6, **B5** = 7, **OUFUKU** = 8, **OFUKU** = 8, **HAGAKI** = 9, **A6_WITH_TEAR_OFF_TAB** = 10, **A3** = 11, **A5** = 12, **LEDGER** = 13, **SUPERB_SIZE** = 14, **EXECUTIVE** = 15, **FLSA** = 16, **CUSTOM_SIZE** = 17, **ENVELOPE_NO_10** = 18, **ENVELOPE_A2** = 19, **ENVELOPE_C6** = 20, **ENVELOPE_DL** = 21, **ENVELOPE_JPN3** = 22, **ENVELOPE_JPN4** = 23, **PHOTO_5x7**, **CDDVD_80**, **CDDVD_120**, **PHOTO_4x8**, **PHOTO_4x12**, **L**, **MAX_PAPER_SIZE** }

Supported Paper sizes.

- enum **TEXTCOLOR** { **WHITE_TEXT**, **CYAN_TEXT**, **MAGENTA_TEXT**, **BLUE_TEXT**, **YELLOW_TEXT**, **GREEN_TEXT**, **RED_TEXT**, **BLACK_TEXT** }

  Supported Text colors for device text.

- enum **COLORMODE** { **GREY_K**, **GREY_CMY**, **COLOR** }

  Color modes for SelectPrintMode.

- enum **QUALITY_MODE** { **QUALITY_NORMAL**, **QUALITY_DRAFT**, **QUALITY_BEST**, **QUALITY_HIGHRES_PHOTO**, **QUALITY_FASTDRAFT**, **QUALITY_AUTO**, **QUALITY_FASTNORMAL** }

  Quality modes for SelectPrintMode.

- enum **MEDIATYPE** { **MEDIA_PLAIN**, **MEDIA_PREMIUM**, **MEDIA_PHOTO**, **MEDIA_TRANSPARENCY**, **MEDIA_HIGHRES_PHOTO**, **MEDIA_AUTO**, **MEDIA_ADVANCED_PHOTO**, **MEDIA_CDDVD** = 7, **MEDIA_BROCHURE** = 8 }

  Media types for SelectPrintMode.

- enum **PHOTOTRAY_STATE** { **UNKNOWN** = -1, **DISENGAGED** = 0, **ENGAGED** = 1 }

  PhotoTray status.

- enum **DRIVER_ERROR** { **NO_ERROR** = 0x00, **JOB_CANCELED** = 0x01, **SYSTEM_ERROR** = 0x02, **ALLOCMEM_ERROR** = 0x03, **NO_PRINTER_SELECTED** = 0x04, **INDEX_OUT_OF_RANGE** = 0x05, **ILLEGAL_RESOLUTION** = 0x06, **NULL_POINTER** = 0x07, **MISSING_PENS** = 0x08, **UNSUPPORTED_PRINTER** = 0x10, **UNSUPPORTED_PEN** = 0x11, **TEXT_UNSUPPORTED** = 0x12, **GRAPHICS_UNSUPPORTED** = 0x13, **UNSUPPORTED_FONT** = 0x14, **ILLEGAL_COORDS** = 0x15, **UNSUPPORTED_FUNCTION** = 0x16, **BAD_INPUT_WIDTH** = 0x18, **OUTPUTWIDTH_EXCEEDS_PAGEWIDTH**

= 0x19, **UNSUPPORTED_SCALING** = 0x1a, **IO_ERROR** = 0x20, **BAD_DEVICE_ID** = 0x21, **CONTINUE_FROM_BLOCK** = 0x22, **PLUGIN_LIBRARY_MISSING** = 0x30, **WARN_MODE_MISMATCH** = -1, **WARN_DUPLEX** = -2, **WARN_LOW_INK_BOTH_PENS** = -3, **WARN_LOW_INK_BLACK** = -4, **WARN_LOW_INK_COLOR** = -5, **WARN_LOW_INK_PHOTO** = -10, **WARN_LOW_INK_GREY** = -11, **WARN_LOW_INK_BLACK_PHOTO** = -12, **WARN_LOW_INK_COLOR_PHOTO** = -13, **WARN_LOW_INK_GREY_PHOTO** = -14, **WARN_LOW_INK_COLOR_GREY** = -15, **WARN_LOW_INK_COLOR_GREY_PHOTO** = -16, **WARN_LOW_INK_COLOR_BLACK_PHOTO** = -17, **WARN_LOW_INK_CYAN** = -18, **WARN_LOW_INK_MAGENTA** = -19, **WARN_LOW_INK_YELLOW** = -20, **WARN_LOW_INK_MULTIPLE_PENS** = - 21, **WARN_FULL_BLEED_UNSUPPORTED** = -6, **WARN_FULL_BLEED_3SIDES** = -7, **WARN_FULL_BLEED_PHOTOPAPER_ONLY** = -30, **WARN_FULL_BLEED_3SIDES_PHOTOPAPER_ONLY** = -31, **WARN_ILLEGAL_PAPERSIZE** = -8 , **WARN_INVALID_MEDIA_SOURCE** = -9 }

Functions

- char * **Version** (BOOL bCompressed)

Variables

- const char **VersionStringID** [21] = "04.09.00A_11-06-2009"
  Version information.
- char **DeveloperString** [32] = "[platform-specific information]"

6.1.1 Detailed Description

Definitions for global variables, types, and defines

6.1.2 Enumeration Type Documentation

6.1.2.1 enum COLORMODE

Enumeration values:

GREY_K Use the BLACK pen to print B&W only.

GREY_CMY Use the COLOR pen to print grey scale.

COLOR Use the COLOR pen to print color.

6.1.2.2 enum DRIVER_ERROR

Enumeration values:

NO_ERROR everything okay.

JOB_CANCELED CANCEL chosen by user.

SYSTEM_ERROR something bad that should not have happened.

ALLOCMEM_ERROR failed to allocate memory.

NO_PRINTER_SELECTED indicates improper calling sequence or unidi.

INDEX_OUT_OF_RANGE what it says.

ILLEGAL_RESOLUTION tried to set resolution at unacceptable value.

NULL_POINTER supplied ptr was null.

MISSING_PENS one or more printhead/pen missing.

UNSUPPORTED_PRINTER selected printer-type unsupported in build.

UNSUPPORTED_PEN selected pen-type unsupported.

TEXT_UNSUPPORTED no text allowed in current build, UsePage-Width is false.

GRAPHICS_UNSUPPORTED no graphics allowed in current build.

UNSUPPORTED_FONT font selection failed.

ILLEGAL_COORDS bad (x,y) passed to TextOut.

UNSUPPORTED_FUNCTION bad selection for PerformPrinter-Function.

BAD_INPUT_WIDTH inputwidth is 0 and.

OUTPUTWIDTH_EXCEEDS_PAGEWIDTH inputwidth exceeds printable width.

UNSUPPORTED_SCALING inputwidth exceeds outputwidth, can't shrink output.

IO_ERROR I/O error communicating with printer.

BAD_DEVICE_ID bad or garbled device id from printer.

CONTINUE_FROM_BLOCK continue from blocked state for printers with no buttons.

PLUGIN_LIBRARY_MISSING a required plugin (dynamic) library is missing.

WARN_MODE_MISMATCH printmode selection incompatible with pen, tray, etc.

WARN_DUPLEX duplexer installed; our driver can't use it.

WARN_LOW_INK_BOTH_PENS sensor says pens below threshold.

WARN_LOW_INK_BLACK sensor says black pen below threshold.

WARN_LOW_INK_COLOR sensor says color pen below threshold.

WARN_LOW_INK_PHOTO sensor says photo pen below threshold.

WARN_LOW_INK_GREY sensor says grey pen below threshold.

WARN_LOW_INK_BLACK_PHOTO sensor says black and photo pens below threshold.

WARN_LOW_INK_COLOR_PHOTO sensor says color and photo pens below threshold.

WARN_LOW_INK_GREY_PHOTO sensor says grey and photo pens below threshold.

WARN_LOW_INK_COLOR_GREY sensor says color and grey pens below threshold.

WARN_LOW_INK_COLOR_GREY_PHOTO sensor says color, photo, and grey pens below threshold.

WARN_LOW_INK_COLOR_BLACK_PHOTO sensor says color, photo, and black pens below threshold.

WARN_LOW_INK_CYAN sensor says cyan ink below threshold.

WARN_LOW_INK_MAGENTA sensor says magenta ink below threshold.

WARN_LOW_INK_YELLOW sensor says yellow ink below threshold.

WARN_LOW_INK_MULTIPLE_PENS sensor says more than one pen below threshold.

WARN_FULL_BLEED_UNSUPPORTED device does not support full-bleed printing.

WARN_FULL_BLEED_3SIDES full bleed on only 3 sides.

WARN_FULL_BLEED_PHOTOPAPER_ONLY device only support full-bleed on photo paper.

WARN_FULL_BLEED_3SIDES_PHOTOPAPER_ONLY device only support 3 sided full-bleed on photo paper.

WARN_ILLEGAL_PAPERSIZE papersize illegal for given hardware.

WARN_INVALID_MEDIA_SOURCE media source tray is invalid.

6.1.2.3 enum MEDIATYPE

Enumeration values:

MEDIA_PLAIN Plain paper.

MEDIA_PREMIUM Premium paper - for use with 6xx series.

MEDIA_PHOTO Photo paper - for use with photo quality printers.

MEDIA_TRANSPARENCY Transparency film.

MEDIA_HIGHRES_PHOTO Premium photo paper.

MEDIA_AUTO Printer uses media sense to determine media type.

MEDIA_ADVANCED_PHOTO Advanced photo paper.

MEDIA_CDDVD CD or DVD media.

MEDIA_BROCHURE Glossy brochure paper.

6.1.2.4 enum PAPER_SIZE

Enumeration values:

UNSUPPORTED_SIZE Not supported paper size (also used as mandatory flag).

LETTER 8.5 x 11 in.

A4 210 x 297 mm.

LEGAL 8.25 x 14 in.

PHOTO_SIZE 4x6 Photo with tear-off tab.

A6 105 x 148 mm.

CARD_4x6 4x6 photo/card without tear-off tab.

B4 250 x 353 mm.

B5 176 x 250 mm.

OUFUKU 148 x 200 mm.

OFUKU Misspelled - here for backwards compatibility.

HAGAKI 100 x 148 mm.

A6_WITH_TEAR_OFF_TAB A6 with tear-off tab.

A3 294 x 419.8 mm.

A5 148 x 210 mm.

LEDGER 11 x 17 in.

SUPERB_SIZE 13 x 19 in.

EXECUTIVE 7.25 x 10.5 in.

FLSA 8.5 x 13 in.

CUSTOM_SIZE Custom.

ENVELOPE_NO_10 No. 10 Envelope (4.12 x 9.5 in.).

ENVELOPE_A2 A2 Envelope (4.37 x 5.75 in.).

ENVELOPE_C6 C6 Envelope (114 x 162 mm).

ENVELOPE_DL DL Envelope (110 x 220 mm).

ENVELOPE_JPN3 Japanese Envelope #3 (120 x 235 mm).

ENVELOPE_JPN4 Japanese Envelope #4 (90 x 205 mm).

PHOTO_5x7 5x7 Photo.

CDDVD_80 3 in. CD or DVD.

CDDVD_120 5 in. CD or DVD.

PHOTO_4x8 Panorama 4 in. x 8 in.

PHOTO_4x12 Panorama 4 in. x 12 in.

L Japanese card (3.5 in. x 5 in.).

MAX_PAPER_SIZE Only for array size and loops.

6.1.2.5 enum PEN_TYPE

Enumeration values:

BLACK_PEN Only BLACK pen in the printer.

COLOR_PEN Only COLOR pen in the printer.

BOTH_PENS BLACK & COLOR pens in printer.

MDL_PEN Photo pen in the printer.

MDL_BOTH COLOR and Photo pen in printer.

MDL_AND_BLACK_PENS BLACK and Photo pen in printer.

MDL_BLACK_AND_COLOR_PENS BLACK, COLOR and Photo pen in printer.

GREY_PEN Only GREY pen in the printer.

GREY_BOTH COLOR and GREY pen in the printer.

MDL_AND_GREY_PENS GREY and Photo pen in printer.

MDL_GREY_AND_COLOR_PENS GREY, COLOR, and Photo pen in the printer.

UNKNOWN_PEN New Pen type that we have no knowledge yet.

NO_PEN No pens in the printer.

DUMMY_PEN Not a possible value - used for initialization.

MAX_PEN_TYPE base 0, ending with MDL_BOTH (NOT NO_PEN).

6.1.2.6 enum PHOTOTRAY_STATE

Enumeration values:

UNKNOWN Unknown State.

DISENGAGED Photo Tray is nor engaged.

ENGAGED Photo Tray is engaged.

6.1.2.7 enum QUALITY_MODE

Enumeration values:

QUALITY_NORMAL Normal quality print mode (probably 300x300).

QUALITY_DRAFT Draft print mode - the same or faster than normal.

QUALITY_BEST Probably slower and possible higher resolution.

QUALITY_HIGHRES_PHOTO 1200 dpi - currently 9xxvip, linux only.

QUALITY_FASTDRAFT True draft, 300 dpi - newer VIP printers only.

QUALITY_AUTO Printer selects optimum resolution - 05 and later VIP printers only.

QUALITY_FASTNORMAL Normal quality print mode - faster than Normal.

6.1.2.8 enum TEXTCOLOR

Enumeration values:

WHITE_TEXT White.

CYAN_TEXT Cyan.

MAGENTA_TEXT Magenta.

BLUE_TEXT Blue.

YELLOW_TEXT Yellow.

GREEN_TEXT Green.

RED_TEXT Red.

BLACK_TEXT Black.

6.1.3 Function Documentation

6.1.3.1 char* Version (BOOL *bCompressed*)

Returns a string identifying components and features of the driver build. If bCompressed is TRUE then the string returned is a bit representation string of the build options (ie. speed, scaling, capture, etc...). If bCompressed is FALSE then the string is a textual concatenation of the developer build string, the APDK version string, and string representation of the build options.

This function returns a string identifying components and features of the driver build, such as which printer-models are supported, whether font and scaling support is included, and other information intended to help solve customer problems.

Parameters:

int bCompressed If true, the information is packed into a decimal number that may be decoded by Hewlett-Packard. Otherwise, the string consists of human-readable indicators.

Returns:

char* string representing version information.

Note:

If bCompressed is TRUE then the return string has only build options and no version information.

6.1.4 Variable Documentation

6.1.4.1 char DeveloperString[32] = "[platform-specific information]"

Used for inserting unique information for your platform. Can be edited here or copied to during construction of your derived Services class to change information. Do not change size of the string.

6.2 Build Options

Modules

- **Font Options**
- **Printer Options**

Defines

- #define **APDK_LITTLE_ENDIAN**
- #define **APDK_AUTODUPLEX**
- #define **APDK_EXTENDED_MEDIASIZE**
- #define **APDK_OPTIMIZE_FOR_SPEED**
- #define **APDK_VIP_COLORFILTERING**
- #define **APDK_NO_NAMESPACE**
- #define **APDK_HIGH_RES_MODES**
- #define **APDK_BUFFER_SEND**
- #define **APDK_CAPTURE**

6.2.1 Detailed Description

This exists primarily as documentation of build options. All build options are documented here. The actual defines are only included if documentation is being built. One can read though this file or the documentation to determine which options to include as input to the build process. config.h also sets up internal macros for including /excluding code based on compiler and defined public macros.

Since:
Version 3.00.01

6.2.2 Define Documentation

6.2.2.1 #define APDK_AUTODUPLEX

Compiles in code that allows for auto duplex on printers that support a duplexer. On most inkjet printers, when the duplexer refeeds the paper, the back side will be rotated 180 degrees with respect to the front side, resulting in tabelt mode duplexing (stapled on the short edge). To print duplexed in booklet mode (stapled on the long edge), the application must rotate the whole page image by 180 degrees before sending the rasters. Call PrintContext::RoateteImage-ForBackPage () to determine if the selected printer needs back page rotated.

6.2.2.2 #define APDK_BUFFER_SEND

This turns on output buffering. Without buffering the APDK sends out each byte as it is ready. With buffer send on the APDK buffers up output data and sends it to the printer when the buffer is full. To set the size of the buffer set the SystemServicesa::iSendBufferSize variable in your derived **SystemServices** (p. 33) and define this directive. The default buffer size, if not specified, is 4096 bytes.

6.2.2.3 #define APDK_CAPTURE

Turn on capturing utility for debugging. This creates script files that document order of calls and state of APDK and PCL output.

6.2.2.4 #define APDK_EXTENDED_MEDIASIZE

Compiles in code that allows for additional media size for large resource system

Since:
Version 3.6.0

6.2.2.5 #define APDK_HIGH_RES_MODES

For embedded devices the APDK defaults to lower (normal) resolution print modes. This facilitates sending less data on lower end (low bandwidth / low CPU) devices. On VIP printers this data is scaled up internally so the print quality is good. For high end devices or OSs, such as Linux, defining APDK_HIGH_RES_MODES will cause some VIP printer modes to send higher resolution data to the printer. This will cause higher CPU utilization and use more I/O bandwidth. However, results could be better if the input data is truly more then 300 DPI. If the input data is not significantly higher then 300 DIP then turning on APDK_HIGH_RES_MODES will unncessarily burn CPU cycles and I/O bandwidth.

Note:
Currently this only effects VIP printers.

Since:
Version 3.00.02

6.2.2.6 #define APDK_LITTLE_ENDIAN

Define this if build is for a little endian processor

Since:
Version 2.3.0

6.2.2.7 #define APDK_NO_NAMESPACE

By default all APDK classes, types, variables, etc... belong to the **apdk** (p. 18) namespace. This minimizes the conflicts when integrating with existing code. The code that calls the APDK driver code should include the line "using namespace APDK_NAMESPACE;". If there are specific conflicts with other names then the name should be specifically specified (i.e. **APDK_NAMESPACE::DRIVER_ERROR** (p. 5) or APDK_NAMESPACE::Job, etc...). Define APDK_NO_NAMESPACE to eliminate the namespace definitions and all APDK names will be placed in the global namespace.

Since:
Version 3.00.01

6.2.2.8 #define APDK_OPTIMIZE_FOR_SPEED

This makes several changes to the code path to ensure maximum speed:

- First, speed optimization forces the use of matrix half toning instead of what the print mode requests. Typically matrix is used for draft modes because it is faster and not as good of quality. Speed optimization uses matrix for all print modes.
- Second, speed optimization forces the use of open color matching instead of proprietary color matching. Open is faster and not as good. Proprietary color matching is not included in the open source code.
- Third, it forces the use of open scaling which is quicker and not as good as proprietary scaling. Proprietary scaling is not include in the open source code.

Since:
Version 2.4.0

6.2.2.9 #define APDK_VIP_COLORFILTERING

By default color filtering (Ernie code) is turned off. Color filtering helps with RLE compression (Bert) when sending data to the printer. Color filtering is only used with VIP printers and should only be turned on when the DJ9xx-VIP family of printers is supported. Turning on color filtering will help in the situation where there is high CPU resources available and low I/O bandwidth. Color filtering reduces the amount of data sent to the printer by matching colors so that RLE compression is more effective. If CPU resources a low do not turn on color filtering.

Note:
APDK_VIP_COLORFILTERING has no effect unless **APDK_DJ9xxVIP** (p. 17) is also defined.

Since:
Version 3.00.00

6.3 Font Options

Defines

- #define **APDK_CGTIMES**

  Include support for CGTimes font - see **Device Fonts** (p. 72).

- #define **APDK_COURIER**

  Include support for Courier font - see **Device Fonts** (p. 72).

- #define **APDK_LTRGOTHIC**

Include support for Letter Gothic font - see **Device Fonts** (p. 72).

- #define **APDK_UNIVERS**

  Include support for Universe font - see **Device Fonts** (p. 72).

6.3.1 Detailed Description

Define one or more of the font directive to enable device fonts. These need to be defined to support hardware fonts in the printer. Enabling fonts is NOT recommended, however if ASCII text needs to be sent to the printer then define the font(s) needed. Only define the fonts that will be used, each font that is enabled will make the code size larger. See **Device Fonts** (p. 72) for more information about fonts or PrintContext::RealizeFont() to create a font.

6.4 Printer Options

Defines

- #define **APDK_APOLLO2XXX**
- #define **APDK_APOLLO21XX**
- #define **APDK_APOLLO2560**
- #define **APDK_DJ350**
- #define **APDK_DJ600**
- #define **APDK_DJ630**
- #define **APDK_DJ6xx**
- #define **APDK_DJ6xxPhoto**
- #define **APDK_DJ8xx**
- #define **APDK_DJ8x5**
- #define **APDK_DJ9xx**
- #define **APDK_DJ9xxVIP**
- #define **APDK_DJGENERICVIP**
- #define **APDK_PSP100**
- #define **APDK_LJMONO**
- #define **APDK_LJCOLOR**
- #define **APDK_LJJETREADY**
- #define **APDK_LJFASTRASTER**
- #define **APDK_LJZJS_MONO**
- #define **APDK_PSCRIPT**
- #define **APDK_DJ3320**
- #define **APDK_DJ3600**
- #define **APDK_LINUX**
- #define **APDK_LDL_COMPRESS**

6.4.1 Detailed Description

Printer families: Define one or more of these to support printer families.

6.4.2 Define Documentation

6.4.2.1 #define APDK_APOLLO21XX

enables support for the Apollo P-2100.

6.4.2.2 #define APDK_APOLLO2560

Enables support for the Apollo P-2500 and P-2600.

6.4.2.3 #define APDK_APOLLO2XXX

Enable support for Apollo P-2200 and the Deskjet 656.

6.4.2.4 #define APDK_DJ3320

Enables support for DJ 3300, 3400, 3500, 3740 family, psc 1100 series, psc 1200 series, officejet 4100 and 4200.

6.4.2.5 #define APDK_DJ350

Enables support for the Deskjet 350.

6.4.2.6 #define APDK_DJ3600

Enables support for DJ 3600, 3840 family, psc 1300 series, and officejet 5500 series.

6.4.2.7 #define APDK_DJ600

Enables support for the Deskjet 600.

6.4.2.8 #define APDK_DJ630

Enables support for the Deskjet 630.

6.4.2.9 #define APDK_DJ6xx

Enables support for the Deskjet 660, 670, 680, 6xx, e-printer.

6.4.2.10 #define APDK_DJ6xxPhoto

Enables support for the Deskjet 610, 640, 690

6.4.2.11 #define APDK_DJ8x5

Enables support for the Deskjet 825, 845. APDK_DJ8xx must also be defined

6.4.2.12 #define APDK_DJ8xx

Enables support for the Deskjet 810, 830, 840, 880, 895, 1125.

6.4.2.13 #define APDK_DJ9xx

Enables support for the Deskjet 920, 930, 94x, 950, 970, 1220, 3816, 3820, PhotoSmart P1000, P1100.

6.4.2.14 #define APDK_DJ9xxVIP

Enables support for the Deskjet 450, 960, 980, 990, 995, 6122, 6127, PhotoSmart 1115, 1215, 1218, 1315 CP 1160, CP 1700 hp business inkjet 2200 series, hp business inkjet 1100 OfficeJet Pro K550, OfficeJet Pro K850 - Note: these two are in OJProKx50 class

6.4.2.15 #define APDK_DJGENERICVIP

Enables support for deskjet 5550, 5551 - Note: these two printers are in DJ55xx class PhotoSmart 7150, 7350, 7550. APDK_DJ9xxVIP must also be defined

6.4.2.16 #define APDK_LDL_COMPRESS

Enables printer ready data compression for DJ3320 and DJ3600 class of printers. This is beneficial where I/O bandwidth is low. May reduce performance on devices that have low power CPUs.

6.4.2.17 #define APDK_LINUX

Enables support for specific Linux resolution requirements for 600 Series

6.4.2.18 #define APDK_LJCOLOR

Enables support for the non-hostbased Color LaserJets.

6.4.2.19 #define APDK_LJFASTRASTER

Enables support for the FastRaster host based LaserJets.

6.4.2.20 #define APDK_LJJETREADY

Enables support for the JetReady host based Color LaserJets.

6.4.2.21 #define APDK_LJMONO

Enables support for the non-hostbased Monochrome LaserJets.

6.4.2.22 #define APDK_LJZJS_MONO

Enables support for LaserJet 1000, 1005, 1018, 1020

6.4.2.23 #define APDK_PSCRIPT

Enables support for the hp and non-hp Postscript Printers

6.4.2.24 #define APDK_PSP100

Enables support for the PhotoSmart 100, 130, 230, 240. APDK_DJ9xxVIP must also be defined

7 APDK Developer's Guide Namespace Documentation

7.1 apdk Namespace Reference

7.1.1 Detailed Description

All classes, types, global variables, functions, etc... are in the apdk namespace to avoid collision with existing code the APDK is integrated with.

Since:
version 3.00.01

8 APDK Developer's Guide Class Documentation

8.1 Font Class Reference

Base class for font support.

Inherited by **ReferenceFont**.

Public Methods

- virtual **DRIVER_ERROR GetTextExtent** (**PrintContext** *pPC, const char *pTextString, const int iLenString, int &iHeight, int &iWidth)

  Calculates width and height of a string using current settings.

- virtual const char * **GetName** () const

  Returns the typeface name.

- virtual BOOL **IsBoldAllowed** () const

  Tells whether text bolding is available.

- virtual BOOL **IsItalicAllowed** () const

  Tells whether text italicizing is available.

- virtual BOOL **IsUnderlineAllowed** () const

  Tells whether text underlining is available.

- virtual BOOL **IsColorAllowed** () const

  Tells whether text coloring is available.

- virtual BOOL **IsProportional** () const

  Tells whether this font is proportionally spaced, as opposed to fixed.

- virtual BOOL **HasSerif** () const

  Tells whether this typeface has serifs.

- virtual BYTE **GetPitch** (const BYTE pointsize) const
- virtual int **Index** ()

  Index of point-size from available list for this font.

Public Attributes

- char **charset** [MAX_CHAR_SET]

8.1.1 Detailed Description

This object is used to control printer fonts when sending ASCII data. (For systems that lack a reasonable font engine or for some reason can benefit from using printer fonts.)

It is not abstract, so that clients can request a font generically, but its constructor is not public – Fonts are rather created through RealizeFont – thus it is effectively "abstract" in this sense. Example: Font* myFont = myJob->RealizeFont(FIXED_SERIF,12,...);

Note that Printer initially constructs a dummy font (with default values) for each of its typefaces, so that the Is<x>Allowed functions can be invoked (for EnumFonts) prior to choosing specific instances. Then the Clone function is invoked by Printer::RealizeFont to provide instances to client.

See also:

ReferenceFont (p. 33)

8.1.2 Member Function Documentation

8.1.2.1 virtual BYTE Font::GetPitch (const BYTE *pointsize*) const [inline, virtual]

For fixed fonts, returns the pitch for given point-size. (Returns zero for proportional fonts.)

8.1.3 Member Data Documentation

8.1.3.1 char Font::charset[MAX_CHAR_SET]

8.2 Job Class Reference

Manages processes to create output for the device.

Public Methods

- **Job** (**PrintContext** *pPC)
- **DRIVER_ERROR SendRasters** (BYTE *ImageData=(BYTE *) NULL)
- **DRIVER_ERROR SendRasters** (BYTE *BlackImageData, BYTE *ColorImageData)
- **DRIVER_ERROR SupportSeparateBlack** (BOOL *bSeparateBlack)
- **DRIVER_ERROR TextOut** (const char *pTextString, unsigned int iLenString, const **Font** &font, int iAbsX, int iAbsY)
- **DRIVER_ERROR Flush** ()
- **DRIVER_ERROR NewPage** ()

8.2.1 Detailed Description

This object receives processes and transmits data to be printed by one device on successive pages. The **PrintContext** (p. 22) and Job together form the core of the driver proper.

See also:
PrintContext (p. 22)

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Job::Job (PrintContext * *pPC*)

constructor - must pass a valid **PrintContext** (p. 22) to create the job. Check constructor_error for NO_ERROR before continuing.

8.2.3 Member Function Documentation

8.2.3.1 DRIVER_ERROR Job::Flush ()

This method forces the associated printer to flush the remaining buffered data and tells it that the job is ended.

Calling this method is not mandatory, since the object destructor already performs this action. It may be required, however, if the destructor is not called for some reason (for instance, when the call depends on the execution of a "finalize" method of a Java wrapper.

8.2.3.2 DRIVER_ERROR Job::NewPage ()

Resets counters, flushes buffers, and sends a form feed to the printer.

8.2.3.3 DRIVER_ERROR Job::SendRasters (BYTE * *BlackImageData*, BYTE * *ColorImageData*)

This is the fundamental method for the driver, by means of which graphics data is sent to the printer.

8.2.3.4 DRIVER_ERROR Job::SendRasters (BYTE * *ImageData* = (BYTE*)NULL)

This is the fundamental method for the driver, by means of which graphics data is sent to the printer.

8.2.3.5 DRIVER_ERROR Job::SupportSeparateBlack (BOOL * *bSeparateBlack*)

This is the method for use to check if they can send a separate 1 bit black channel

8.2.3.6 DRIVER_ERROR Job::TextOut (const char * *pTextString*, unsigned int *iLenString*, const Font & *font*, int *iAbsX*, int *iAbsY*)

This method is used to send ASCII data to the printer, which it will render as determined by the **Font** (p. 18) object.

8.3 PrintContext Class Reference

Allows setting options of a print job based on specific device.

Public Methods

- **PrintContext** (**SystemServices** *pSysServ, unsigned int InputPixelsPerRow=0, unsigned int OutputPixelsPerRow=0, **PAPER_SIZE** ps=LETTER, **QUALITY_MODE** eQuality=QUALITY_NORMAL, **MEDIATYPE** eMedia=MEDIA_PLAIN, **COLORMODE** eColorMode=COLOR, BOOL bDeviceText=FALSE)

  Construct a context for the print device.

- virtual ~**PrintContext** ()

  Destroy the print device context.

- void **Flush** (int FlushSize)
- **DRIVER_ERROR SelectDevice** (const PRINTER_TYPE Model)
- **DRIVER_ERROR SelectDevice** (const char *szDeviceIDString)

  Select device type using device id string returned from printer.

- unsigned int **GetModeCount** ()

  brief Returns number of supported print modes for the select printer model.

- **DRIVER_ERROR SelectPrintMode** (const unsigned int index)
- **DRIVER_ERROR SelectPrintMode** (**QUALITY_MODE** eQuality=QUALITY_NORMAL, **MEDIATYPE** eMedia=MEDIA_PLAIN, **COLORMODE** eColorMode=COLOR, BOOL bDeviceText=FALSE)

  Select a print mode based on independent parameters.

- **DRIVER_ERROR SetPenSet** (**PEN_TYPE** ePen)
- unsigned int **CurrentPrintMode** ()

  Returns the current print mode index.

- const char * **GetModeName** ()
- **DRIVER_ERROR GetPrintModeSettings** (**QUALITY_MODE** &eQuality, **MEDIATYPE** &eMedia, **COLORMODE** &eColorMode, BOOL &bDeviceText)

  Returns pointer to print mode settings.

- PRINTER_TYPE **SelectedDevice** ()
- PRINTER_TYPE **EnumDevices** (FAMILY_HANDLE &familyHandle) const
- **DRIVER_ERROR PerformPrinterFunction** (PRINTER_FUNC eFunc)
- **DRIVER_ERROR SetPaperSize** (**PAPER_SIZE** ps, BOOL bFullBleed=FALSE)

  Sets or changes the target paper size for a print job.

- **DRIVER_ERROR SetPixelsPerRow** (unsigned int InputPixelsPerRow, unsigned int OutputPixelsPerRow=0)
- BOOL **PrinterSelected** ()

  Returns TRUE if printer has been selected.

- BOOL **PrinterFontsAvailable** ()

  Returns TRUE if the selected printer (and the current build) provides font support.

- BOOL **SupportSeparateBlack** ()

  Returns TRUE if the selected printer (and the current build) provides separate 1 bit balck channel.

- unsigned int **InputPixelsPerRow** ()

  Returns the width as set by SetPixelsPerRow.

- unsigned int **OutputPixelsPerRow** ()

  Returns the current setting for output width.

- **PAPER_SIZE GetPaperSize** ()

  Returns the currently set paper size.

- BOOL **PhotoTrayPresent** (BOOL bQueryPrinter)

  Returns TRUE if a photo tray is present in printer.

- **PHOTOTRAY_STATE PhotoTrayEngaged** (BOOL bQueryPrinter)

  Returns current state of phototray, one of UNKNOWN, DISENGAGED or ENGAGED.

- BOOL **HagakiFeedPresent** (BOOL bQueryPrinter)

  Returns TRUE if a hagaki feed is present in printer.

- BOOL **HagakiFeedDuplexerPresent** (BOOL bQueryPrinter)

  Returns TRUE if duplexer and hagaki feed (combined) unit is present in printer.

- const char * **PrinterModel** ()
- const char * **PrintertypeToString** (PRINTER_TYPE pt)
- unsigned int **EffectiveResolutionX** ()

  Returns the horizontal printer resolution to be used for currently selected print mode.

- unsigned int **EffectiveResolutionY** ()

  Returns the vertical printer resolution to be used for currently selected print mode.

- float **PrintableWidth** ()

  Returns width of printable region in inches.

- float **PrintableHeight** ()

  Returns height of printable region in inches.

- float **PhysicalPageSizeX** ()

  Retrieves information about the physical width of the currently selected paper size.

- float **PhysicalPageSizeY** ()

  Retrieves information about the physical height of the currently selected paper size.

- float **PrintableStartX** ()

  Returns the left margin or distance from the top edge of the page.

- float **PrintableStartY** ()

  Returns the top margin or distance from the top edge of the page.

- **DRIVER_ERROR SendPrinterReadyData** (BYTE *stream, unsigned int size)

- **DRIVER_ERROR SetMediaSource** (MediaSource num)

  Select input media source bin.

- MediaSource **GetMediaSource** ()

  Return input media source bin.

- BOOL **SelectDuplexPrinting** (DUPLEXMODE duplexmode)

  Set two-sided printing option.

- DUPLEXMODE **QueryDuplexMode** ()

  Returns current setting for two-sided printing option.

- BOOL **RotateImageForBackPage** ()

  Does back page needs to be rotated?

- unsigned int **GetCurrentDyeCount** ()

  Get number of color markers that will used.

- **PEN_TYPE GetDefaultPenSet** ()

  Get the default pen combination for this printer.

- **PEN_TYPE GetInstalledPens** ()

  Get ink cartridges/toners currently installed.

- void **ResetIOMode** (BOOL bDevID, BOOL bStatus)

  Change bi-di communication mode.

- int **GetCopyCount** ()

  Get number of copies to print.

- void **SetCopyCount** (int iNumCopies)

  Set number of copies to print.

- **DRIVER_ERROR SetPrinterHint** (PRINTER_HINT eHint, int iValue)

  Request certain printer specific functionality.

- BOOL **IsBorderless** ()

  Returns TRUE if borderless printing is enabled, FALSE otherwise.

- **DRIVER_ERROR SetMediaType** (**MEDIATYPE** eMediaType)

  SetMediaType.

- void **SetMediaSubtype** (int iMediaSubtype)

  SetMediaSubType.

- void **SetMechOffset** (int iMechOffset)

  Set the firmware Offset for Top (TopMargin) in Inch∗1000.

Friends

- class **Job**

8.3.1 Detailed Description

This object serves two purposes. First, it encapsulates knowledge about all devices supported in available driver components. It can be queried at runtime to determine capabilities of the attached device. Second, it allows for setting optional parameters of a print job, such as media size and margins, use of color and data-resolution, etc.

The PrintContext is the second item created in the driver calling sequence, following **SystemServices** (p. 33). It provides the interface for inquiring about supported devices, and for setting all optional properties of the imaging pipeline as determined by given devices.

See also:
Job (p. 20) **SystemServices** (p. 33)

8.3.2 Constructor & Destructor Documentation

8.3.2.1 PrintContext::PrintContext (SystemServices ∗ *pSysServ*, unsigned int *InputPixelsPerRow* = 0, unsigned int *OutputPixelsPerRow* = 0, PAPER_SIZE *ps* = LETTER, QUALITY_MODE *eQuality* = QUALITY_NORMAL, MEDIATYPE *eMedia* = MEDIA_PLAIN, COLORMODE *eColorMode* = COLOR, BOOL *bDeviceText* = FALSE)

In the normal case where bidirectional communication was established by **SystemServices** (p. 33), successful construction results in instantiation of the proper Printer class; otherwise client will have to complete the process with a subsequent call to SelectDevice. The next two parameters are optional. In the case where InputPixelsPerRow has the default setting of zero, the value of printablewith will be used. If the desired print mode is known in advance, (i.e. The user won't be selecting a print mode at any point) and the image size is also

known, then the print mode can be selected at during construction. The reason why the last four parameters were added was because if InputPixelsPerRow and OutputPixelsPerRow were in a resolution greater than 300 dpi, it might cause the constructor to set constructor_error to ILLEGAL_COORDS. The constructor checks to see if the OutputPixelsPerRow will fit on the default print mode's page width (via a call to SelectPrintMode where the real check is done), and if it doesn't fit, the error was given even if it was still a legal output size in the resolution. Using the last four parameters will ensure that the image size is checked using the proper print mode.

Parameters:

pSysServ Your previously created **SystemServices** (p. 33)

InputPixelsPerRow Input pixel witdth per row (do not exceed pagewidth ∗ resolution)

OutputPixelsPerRow Usually set to 0 (zero) for scaling up to pagewidth

ps Paper size that will be used

eQuality Quality of output: DRAFT, NORMAL, BEST (default NORMAL)

eMedia Media type: PLAIN, PREMIUM, PHOTO (default PLAIN)

eColorMode Color Mode: GREY_K, GREY_CMY, COLOR (default COLOR)

bDeviceText Support Device Text: TRUE, FALSE (dafault FALSE)

8.3.3 Member Function Documentation

8.3.3.1 unsigned int PrintContext::CurrentPrintMode ()

Deprecated:

Retrieves an index of the currently selected print mode.

8.3.3.2 PRINTER_TYPE PrintContext::EnumDevices (FAMILY_HANDLE & *familyHandle*) const

Returns the enum for the next supported model. This method is used when bi-directional communication is missing, or for dynamic querying of the properties of a given build. This method is presumably called within a loop, which will exit when the return value equals UNSUPPORTED. Passing this return value to SelectDevice will instantiate this printer object and therefore allow further querying of the printers capabilities through other methods in PrintContext.

Parameters:

familyHandle Caller starts at null; reference variable is incremented automatically for next call.

Returns:

A value matching an element of the enum PRINTER_TYPE, which can then be passed to other methods such as SelectDevice.

8.3.3.3 void PrintContext::Flush (int *FlushSize*)

Used outside the context of a job. Flushes data in the printer's input buffer to prepare for new data stream.

8.3.3.4 unsigned int PrintContext::GetCurrentDyeCount ()

This method returns the number of coloring agents (ink/toner) that will be used to print this job with the selected printmode.

8.3.3.5 PEN_TYPE PrintContext::GetDefaultPenSet ()

This method returns the number of ink cartridges/toners that will be used by default by this printer. Where permitted, SetPens can be used to change the current selection. See PEN_TYPE enum in global_types.h for possible pen combinations.

8.3.3.6 PEN_TYPE PrintContext::GetInstalledPens ()

This method returns the set of ink cartridges currently installed. This information is valid only if bi-directional I/O is supported, otherwise, it will return the default pen set required by the printer.

8.3.3.7 MediaSource PrintContext::GetMediaSource () `[inline]`

Used to get the bin number from which media will be loaded by the printer. This is relevant for those printers that have multiple input bins. All other printers will ignore the bin number. The typical bin numbers are 1 - Upper Tray 4 - Lower Tray 7 - Auto Select Any value between 1 and 50 is valid where there are more than 2 trays.

8.3.3.8 unsigned int PrintContext::GetModeCount ()

Deprecated:

This method will return the number of print modes. The device must support (and should return a value of) at least two print modes, gray and normal. The general convention for interpretation of the indices is:

- 0 = GrayMode - rendering for black pen
- 1 = the \i basic mode for this printer, usually targeting plain paper and normal quality

- 2,3... = special modes that may be available for this printer

Note:
Do not count on mode counts or mode indexes.

See also:
SelectPrintMode() (p. 31)

8.3.3.9 const char* PrintContext::GetModeName () [inline]

Deprecated:
Returns the identifying string for the currently selected print mode. This method now returns an empty string for every print mode.

8.3.3.10 DRIVER_ERROR PrintContext::GetPrintModeSettings (QUALITY_MODE & *eQuality*, MEDIATYPE & *eMedia*, COLORMODE & *eColorMode*, BOOL & *bDeviceText*)

Used to determine the currently selected print mode. Print mode could be different then requested values in SelectePrintMode because the printer may not be able to support the requested combination. In that case the PrintContext will make an intelegent choice about changing the print mode. This method can be used to determin the differences between the requested mode and the selected mode.

Parameters:
eQuality Quality of output: DRAFT, NORMAL, BEST

eMedia Media type: PLAIN, PREMIUM, PHOTO

eColorMode Color Mode: GREY_K, GREY_CMY, COLOR

bDeviceText Support Device Text: TRUE, FALSE

8.3.3.11 DRIVER_ERROR PrintContext::PerformPrinterFunction (PRINTER_FUNC *eFunc*)

Used outside the context of a **Job** (p. 20), to perform functions such as pen cleaning.

8.3.3.12 const char * PrintContext::PrinterModel ()

Returns the model portion of the firmware ID string from the printer.

8.3.3.13 const char * PrintContext::PrintertypeToString (PRINTER_TYPE *pt*)

Returns a string representing the printer model family signified by the parameter.

8.3.3.14 DUPLEXMODE PrintContext::QueryDuplexMode ()

Determine what duplexing mode is currently selected.

8.3.3.15 void PrintContext::ResetIOMode (BOOL *bDevID*, BOOL *bStatus*)

Use this method to change bi-di communication mode. When bi-di I/O is possible, intended use is to allow bi-di initially, so APDK can instantiate the correct printer class and then reset to uni-di mode so the front end can manage status handling.

8.3.3.16 BOOL PrintContext::RotateImageForBackPage ()

When automatic two-sided printing is enabled for long edge binging, application may have to rotate the back page by 180 degrees.

8.3.3.17 DRIVER_ERROR PrintContext::SelectDevice (const PRINTER_TYPE *Model*)

Used by client when **SystemServices** (p. 33) couldn't get DevID. This is the place where printer gets instantiated for unidi

8.3.3.18 BOOL PrintContext::SelectDuplexPrinting (DUPLEXMODE *duplexmode*)

This has no effect on printers that do not support an automatic duplexer mechanism. Two sided-printing option can be one of DUPLEXMODE_NONE - one sided printing DUPLEXMODE_TABLET - binding on short edge DUPLEXMODE_BOOK - binding on long edge

8.3.3.19 PRINTER_TYPE PrintContext::SelectedDevice ()

This method returns the currently selected device. A device is selected implicitly in the constructor if bi-directional communication is working. **SelectDevice**() (p. 30) is used with unidirectional communication, or to override the implicit selection.

Returns:

PRINTER_TYPE Returns UNSUPPORTED if no Printer has been selected, either implicitly or explicitly.

8.3.3.20 DRIVER_ERROR PrintContext::SelectPrintMode (QUALITY_MODE *eQuality* = QUALITY_NORMAL, MEDIATYPE *eMedia* = MEDIA_PLAIN, COLORMODE *eColorMode* = COLOR, BOOL *bDeviceText* = FALSE)

Select a print mode base on Quality, Media, ColorMode, and support of DeviceText. Method may change parameters depending on what the printer supports.

See also:

GetPrintModeSettings (p. 29)

Parameters:

eQuality Quality of output: DRAFT, NORMAL, BEST

eMedia Media type: PLAIN, PREMIUM, PHOTO

eColorMode Color Mode: GREY_K, GREY_CMY, COLOR

bDeviceText Support Device Text: TRUE, FALSE

8.3.3.21 DRIVER_ERROR PrintContext::SelectPrintMode (const unsigned int *index*)

Deprecated:

Chooses amongst available print modes. Index of zero is grayscale, index of 1 is default mode. Use the new SelectePrintMode interface.

8.3.3.22 DRIVER_ERROR PrintContext::SendPrinterReadyData (BYTE * *stream*, unsigned int *size*)

Used outside the context of a job. For use when pre-formatted data is available from an external source.

8.3.3.23 void PrintContext::SetCopyCount (int *iNumCopies*) [inline]

Copy count has no effect on printers that do not support this feature. Typically, most LaserJet printers support multiple copies, whereas, Inkjets do not.

8.3.3.24 DRIVER_ERROR PrintContext::SetMediaSource (MediaSource *num*)

Used to set the bin number from which media will be loaded by the printer. This is relevant for those printers that have multiple input bins. All other printers will ignore the bin number. The typical bin numbers are 1 - Upper Tray 4 - Lower Tray 5 - Duplexer Hagaki Feed 6 - Photo Tray for Inkjet devices 7 - Auto Select 14 - CD/DVD tray for Inkjet devices Any value between 1 and 50 is valid where there are more than 2 trays.

Parameters:

num Bin Number

8.3.3.25 void PrintContext::SetMediaSubtype (int *iMediaSubtype*) [inline]

The combination of MediaType and MediaSubType select the exact media type to use for printing. There is no dependency for MediaSubType with other attributes. The iMediaSubtype value passed is sent to printer using the Driver Function Configuration command. Ex: For Glossy Brochure, Media Type is 8, Media SubType is 0x460. So 0x460 should be passed to SetMediaSubtype and MEDIA_BROCHURE to SetMediaType

8.3.3.26 DRIVER_ERROR PrintContext::SetMediaType (MEDIATYPE *eMediaType*)

Typically, media type is bound to a print mode and is set when a printmode is selected. This is because the mediatype, printmode combination affects colormatching and requires a different colormap data. Some printers do colormatching in firmware and may support multiple mediatypes for a selected printmode. Returns NO_ERROR if requested mediatype is supported for the selected printmode, otherwise returns WARN_PRINTMODE_MISMATCH and the mediatype remains unchanged in the printmode.

8.3.3.27 DRIVER_ERROR PrintContext::SetPaperSize (PAPER_SIZE *ps*, BOOL *bFullBleed* = FALSE)

This method sets the target paper size to be used by the print job. This would have already been set during PrintContext construction, but may be reset using this function as long as the job object itself has not yet been created.

See also:

PAPER_SIZE (p. 7) for supported paper sizes.

8.3.3.28 DRIVER_ERROR PrintContext::SetPenSet (PEN_TYPE *ePen*)

Sets the pen set to be used (much match what is actually in the printer). For use in uni-di mode.

8.3.3.29 DRIVER_ERROR PrintContext::SetPixelsPerRow (unsigned int *InputPixelsPerRow*, unsigned int *OutputPixelsPerRow* = 0)

Sets the width of all rows on the page.

8.3.3.30 DRIVER_ERROR PrintContext::SetPrinterHint (PRINTER_HINT *eHint*, int *iValue*)

Some printers have non-standard options. This method allows the application to request such options. This request is passed on to the printer class which handles what options it can support. You must know the printer's internal specifications to use this method. Supported hints are: PAGES_IN_DOC_HINT - set total number of pages in the print job SPEED_MECH_HINT - enable faster throughput by picking next page as the current page advances EXTRA_DRYTIME_HINT - wait for additional seconds to let page dry in duplex jobs MAX_FILE_SIZE_HINT - set maximum size of jpeg file produced by quickconnect printer RED_EYE_REMOVAL_HINT - set red eye removal option for quickconnect printer PHOTO_FIX_HINT - set photo fix option for quickconnect printer LEFT_OEVERSPRAY_HINT - set left overspray value for borderless job in inch * 1000 TOP_OVERSPRAY_HINT - set top overspray value for borderless job in inch * 1000 RIGHT_OVERSPRAY_HINT - set right overspray value for borderless job in inch * 1000 BOTTOM_OVERSPRAY_HINT - set bottom overspray value for borderless job in inch * 1000

8.3.3.31 BOOL PrintContext::SupportSeparateBlack ()

This is the method for use to check if the printer support a separate 1 bit black channel

8.4 ReferenceFont Class Reference

Used by **Job** (p. 20) to realize a font.

Inherits **Font**.

8.4.1 Detailed Description

Subclass ReferenceFont (EnumFont) is used to query available font properties prior to instantiating the font. Whereas **Font** (p. 18) objects created upon request are to be deleted by caller, ReferenceFonts live with the core structures and cannot be deleted.

The main purpose of this class is to hide the destructor, since the fonts that live with the Printer and are returned by EnumFont are meant to remain alive for the life of the Printer.

See also:
Font (p. 18) Printer

8.5 SystemServices Class Reference

Provides interface to host environment.

Public Methods

- **SystemServices** ()
- virtual ~**SystemServices** ()
- **DRIVER_ERROR InitDeviceComm** ()

  Must include in derived class constructor (if using bi-di).

- virtual **DRIVER_ERROR FlushIO** ()

  Passes a command to the I/O system to send all data to the printer immediately.

- virtual **DRIVER_ERROR AbortIO** ()

  Tells the I/O system that the print job has ended prematurely.

- virtual void **DisplayPrinterStatus** (DISPLAY_STATUS ePrinterStatus)=0
- virtual **DRIVER_ERROR BusyWait** (DWORD msec)=0
- virtual **DRIVER_ERROR ReadDeviceID** (BYTE *strID, int iSize)=0
- virtual BYTE * **AllocMem** (int iMemSize)=0
- virtual void **FreeMem** (BYTE *pMem)=0
- virtual BOOL **PrinterIsAlive** ()
- virtual BOOL **GetStatusInfo** (BYTE *bStatReg)=0
- virtual **DRIVER_ERROR ToDevice** (const BYTE *pBuffer, DWORD *dwCount)=0
- virtual **DRIVER_ERROR FromDevice** (BYTE *pReadBuff, DWORD *wReadCount)=0

- virtual **DRIVER_ERROR GetECPStatus** (BYTE *pStatusString, int *pECPLength, int ECPChannel)
- virtual DWORD **GetSystemTickCount** (void)=0
- const char * **PrinterModel** ()
- unsigned int **GetSendBufferSize** () const
- **DRIVER_ERROR FreeMemory** (void *ptr)
- **DRIVER_ERROR GetDeviceID** (BYTE *strID, int iSize, BOOL query)
- int **GetVIPVersion** ()

Public Attributes

- **DRIVER_ERROR constructor_error**

Protected Methods

- virtual void **AdjustIO** (IO_MODE IM, const char *model=NULL)

  reconcile printer's preferred settings with reality.

Friends

- class **PrintContext**

8.5.1 Detailed Description

The SystemServices object is used to encapsulate memory-management, I/O, clock access, user notification and UI, and other utilities provided by the operating system or host system supporting the driver. It is an abstract base class, requiring implementation of the necessary routines by the host. Creation of SystemServices is the first step in the calling sequence for the driver, followed by creation of the **PrintContext** (p. 22) and the **Job** (p. 20). The derived class constructor must include a call to the member function InitDeviceComm, which establishes communication with the printer if possible. The only reference to this object in API-calling code should be to its constructor and destructor, which must be invoked prior to and after all other driver calls.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 SystemServices::SystemServices ()

This method constructs a SystemServices object. This is the first step in the calling sequence. Check constructor_error before continuing.

8.5.2.2 SystemServices::~SystemServices () [virtual]

This method is a destructor, called when the instance of SystemServices is deleted or goes out of scope.

8.5.3 Member Function Documentation

8.5.3.1 virtual DRIVER_ERROR SystemServices::AbortIO () [inline, virtual]

This routine instructs the I/O system that this print job has ended prematurely, that no further data will be sent and that any data being held in a "send" buffer can be thrown away. This command is optional and depends on the implementation of the I/O system. In some error conditions, the printer is incapable of accepting more data. The printer driver will sense this error and stop sending data, but if the I/O system has buffered data it may continue to wait for the printer to accept this remaining data before unloading. This mechanism will allow for a more efficient clean up of the aborted job.

8.5.3.2 virtual BYTE* SystemServices::AllocMem (int *iMemSize*) [pure virtual]

This method will allocate a contiguous block of memory at least iMemSize bytes long.

Parameters:

iMemSize - size of block.

Returns:

Pointer to the memory block. Can also return NULL, if allocation failed.

8.5.3.3 virtual DRIVER_ERROR SystemServices::BusyWait (DWORD *msec*) [pure virtual]

Must be implemented in derived class. Waits for the specified number of milliseconds before returning to the caller. Gives host system a chance to process asynchronous events. Derived implimentation of BusyWait MUST allow the processing of asynchronous events such as user interface changes, response to dialogs or pop-ups. The BusyWait method should return NO_ERROR if we should continue processing. The APDK core code calls busy wait within loops (when the printer is out of paper, has no pen, etc...) and will continue when the user has fixed the problem. If the user wants to cancel the job then the implemented version of BusyWait must return JOB_CANCELED so that the APDK stops retrying and cancels the job.

8.5.3.4 virtual void SystemServices::DisplayPrinterStatus (DISPLAY_STATUS *ePrinterStatus*) [pure virtual]

Uses message code to relay a message to the user. Must be implemented in derived class provided by developer. This routine will cause a corresponding text string (or graphic, or both) to be displayed to the user to inform them of printer status. This is how printer events such as out-of-paper will be communicated to the user.

8.5.3.5 virtual DRIVER_ERROR SystemServices::FlushIO () [inline, virtual]

A typical I/O system either buffer data or send data immediatly. If the I/O system is buffering data then typically store all data sent in a buffer until it has some amount of data (say 1, 2, or 4 Kbytes) then will send that block of data to the printer. In some cases, it may be necessary that data that has just been sent to the I/O system reach the printer ASAP, at which point the printer driver will check the status or deviceID string of the printer and continue. Often, USB implementations will favor the block-send mechanism because it is more efficient through the system's call stack. Simple and traditional centronics I/O is usually not buffered, but this is still a system-by-system I/O preference.

If the I/O system is buffering data into blocks before sending to the printer, this command should pass a command to the I/O system to stop waiting and send everything it has to the printer now. The base implimentation simply returns NO_ERROR and is appropriate for a system that does not buffer I/O.

Note:

The appropriate way to handle a FlushIO on a buffered system is to call the USB class driver's I/O Control routine with a flush command. If the USB class driver has no flush command then check to see if it has timed flush (it auto-flushes every n [milli]seconds). If it does then the base FlushIO method will work okay.

Warning:

In the past some have implemented flush by sending 10K nulls to the USB driver. Currently this does work and will cause data already in the buffer to be sent to the printer. Current printers ignore the null, however, there may be printers in the future that cannot handle this behavior.

8.5.3.6 virtual void SystemServices::FreeMem (BYTE * *pMem*) [pure virtual]

Frees a given block of memory previously allocated with AllocMem.

Parameters:

pMem - pointer to the block of memory to be freed.

8.5.3.7 DRIVER_ERROR SystemServices::FreeMemory (void * *ptr*)

Same as host-supplied FreeMem, with extra safety check for null pointer.

8.5.3.8 virtual DRIVER_ERROR SystemServices::FromDevice (BYTE * *pReadBuff*, DWORD * *wReadCount*) [pure virtual]

Gets data from printer.

Parameters:

pReadBuff Pointer to buffer into which data is to be copied.

wReadCount Output parameter (pointer to allocated DWORD) telling number of bytes copied.

8.5.3.9 DRIVER_ERROR SystemServices::GetDeviceID (BYTE * *strID*, int *iSize*, BOOL *bQuery*)

This method tries to get the device id back from the printer and does some basic verification.

8.5.3.10 DRIVER_ERROR SystemServices::GetECPStatus (BYTE * *pStatusString*, int * *pECPLength*, int *ECPChannel*) [virtual]

This function will open an ECP I/O channel to the printer and retrieve a given number of bytes from it. Because ECP is a 1284 protocol, this function is only relevant for 1284 compliant parallel I/O connectivity. Currently, only the DeskJet 4xx Series of printers requires implementation of this function. Because of the non-standard nature of this function, it is expected that the sample code supplied in ECPSample.cpp will be heavily leveraged during implementation of this function in the derived SystemServices class.

Note:

The DJ400 & DJ540 code was written as a special for a specific host. DJ400 & DJ500 are not supported by the APDK and there is no help with communication issues with these printers.

Parameters:

pStatusString The destination of the set of retrieved status bytes.

pECPLength The number of retrieved bytes.

ECPChannel The ECP channel number to be opened and read.

8.5.3.11 unsigned int SystemServices::GetSendBufferSize () const [inline]

Returns the size of the send buffer.

8.5.3.12 virtual BOOL SystemServices::GetStatusInfo (BYTE * *bStatReg*) [pure virtual]

This method reads status register from printer. If the parallel status byte is returned, the function returns TRUE when it sets this value. Otherwise the function returns FALSE if this functionality is not supported.

Parameters:

bStatReg Pointer to a byte into which status register will be copied.

Returns:

TRUE The value is set after returning the parallel status byte, FALSE if the functionality is not supported.

8.5.3.13 virtual DWORD SystemServices::GetSystemTickCount (void) [pure virtual]

This routine will query the system for the current tick count (time). Since it will be used primarily for retry time-outs in the driver, it only needs to provide a running count of elapsed time since the device was booted, for example.

Returns:

DWORD Number of ticks for current tick count.

8.5.3.14 int SystemServices::GetVIPVersion () [inline]

Return the VIP version of the firmware ID from the printer.

8.5.3.15 DRIVER_ERROR SystemServices::InitDeviceComm ()

Mandatory call to be inserted in derived constructor. This method tries to establish communications with printer and identify it. The derived System-Services constructor must call this base-class routine.

8.5.3.16 BOOL SystemServices::PrinterIsAlive () [virtual]

Tests communication with printer. It calls host supplied GetStatusInfo.

Returns:

TRUE if communication with printer is working.

Note:
This implementation is appropriate for Parallel bus only.

8.5.3.17 const char* SystemServices::PrinterModel () [inline]

Returns the model portion of the firmware ID string from the printer.

8.5.3.18 virtual DRIVER_ERROR SystemServices::ReadDeviceID (BYTE * *strID*, int *iSize*) [pure virtual]

Retrieves the device identifier string from the printer.

Parameters:
strID Pointer to buffer for storing string.

iSize The size of buffer.

Returns:
IO_ERROR

8.5.3.19 virtual DRIVER_ERROR SystemServices::ToDevice (const BYTE * *pBuffer*, DWORD * *dwCount*) [pure virtual]

Sends bytes of data to the printer.

Note:
This method decrements the dwCount parameter (*dwCount) by the number of bytes that were actually sent.

Parameters:
pBuffer pointer to buffer full of data.

dwCount (in/out) Upon entry, this contains the number of bytes of data requested to send. Upon return from the method, dwCount is the number of bytes that were not sent, i.e. the remaining bytes.

8.5.4 Member Data Documentation

8.5.4.1 DRIVER_ERROR SystemServices::constructor_error

Check this member variable for the validity of the constructed object. Calling code must check that the value of this variable is NO_ERROR before proceeding.

See also:
DRIVER_ERROR (p. 5)

9 APDK Developer's Guide Page Documentation

9.1 Developer's Guide

9.2 APDK license

9.3 Introduction

Welcome to the Hewlett-Packard Appliance Printing Software Development Kit (APDK). This kit contains everything you need to begin developing a Hewlett-Packard DeskJet printer driver for your host system. The APDK is comprised of documentation, sample test and HTML help files, tools to aid the developer in integrating and testing their printer driver, and most importantly, core source code written in C++.

The APDK contains code that, although not a printer driver by itself, will be the basis for creating a very small yet functional printer driver. This code contains space-optimized versions of many of the same core imaging technologies used in full-featured Hewlett-Packard Windows drivers, as well as basic print mode support and built-in printer text support.

The APDK does not assume that developers have any direct knowledge of printer drivers; It does assume that programmers have a working knowledge of C and C++, including the principles and vocabulary of object oriented design. The developer must be familiar with all aspects of their own host system, which is needed to render and format data in a manner suitable for the printed page.

The APDK manual is divided into several sections:

9.4 Overview

The SDK final product is designed to accept pre-formatted, 24-bit sRGB raster data and/or ASCII text, convert it to a high-quality, printer-ready data stream, and send it to a Hewlett-Packard DeskJet printer via a host-implemented I/O mechanism. The SDK also provides basic structures needed for job control, as well as functions to perform various printer-maintenance activities, (e.g. cleaning the pens).

The SDK is primarily responsible for converting device-independent data into a device-dependent, printer-optimized stream of bytes. Because the conversion process for color inkjet printers is quite complicated, it is important to not change device-independent portions of the SDK code. The color science involved is relatively complex and requires a thorough understanding of Hewlett-Packard's inkjet technology.

The SDK code also interprets status information from the printer and relays it to the system for presentation to the user. Note that the SDK code itself does not attempt to notify or directly interact with the user. Instead user interfaces for initiating or configuring a print job and visual presentation of printing status information are left to creation by the developer. This provides flexibility for the operating systems to maintain a consistent look and feel. Ultimately, this should help insulate developers from most of the details encountered with printing, and instead allow them to focus on other areas.

The developer must realize that the creation of a printer driver is only one piece of the entire printing solution. If the target platform already has existing print architecture, the effort to create and integrate the new driver may be minimal. However, if there is no pre-existing print functionality, or if the functionality is substantially different from what the APDK assumes about the host environments, then additional work will be required.

Additional efforts might include modifying the application software on the target platform to support printing, or adding necessary infrastructure support for optional background printing or spooling. Other documents within this kit will discuss some of these options in greater detail and guide the developer towards a more complete printing solution to satisfy the end customer.

9.5 Getting Started

9.5.1 Design Objectives

This APDK has been developed primarily to meet the demand for Hewlett-Packard DeskJet connectivity in the non-PC markets. Due to the varying nature and diversity of devices that have or might desire printing capabilities, the APDK core code has been created with several key objectives:

- Small driver size - to meet the needs of small memory footprint devices.
- Portability - to enable quick deployment on all kinds of devices.
- Extensibility - to allow for future enhancements.
- Features & Performance - to provide the proper balance of print quality, speed and feature support, given the current size and portability constraints.

Many of these supported features are implemented in such a way that they are both modular and optional. If a given device will not use an available feature, it is okay to be left out. However, the developer is strongly encouraged to communicate the level of support (features, performance, etc.) offered, with their customers. (For example, developers should state for their customers that banners and duplex printing are not supported, regardless of the fact that the printers have the functional capability.) This should be made clear on any appropriate Web pages, on-line help pages, in manuals, etc., in order to help the user establish the proper expectations before purchasing a printer for a specific device or environment, or trying to print in an unsupported manner.

9.5.2 Limitations

In order to achieve each of the APDK design objectives (small driver size, portability, etc.,) the finished printer driver will not, and cannot be as full featured as Hewlett- Packard's corresponding Windows driver. Size and portability are the main restrictions that limit the features provided in this kind of driver. For a complete list of features supported by printer model, refer to **Printers and Features Supported** (p. 86) Printers and Features Supported table.

9.5.3 Minimum Host System Requirements

The minimum host system requirements necessary to operate a printer driver fall into the following basic categories:

Hardware:

- A development system of the developer's own choosing (such as a PC running Windows NT, etc.)
- A supported Hewlett-Packard printer. (See the Printing section's Printers and Features Supported list.)
- Parallel or USB Cable (IEEE STD 1284-1994 compliant recommended).

Software:

- C ++ compiler and linker.
- Standard libraries supporting string and other functions

The APDK code has been built and tested using both GNU and Microsoft compilers.

Development Requirements:

- C++ Compiler. (Developers can create the appropriate binary printer driver by using compilers and tools of their own choosing (such as GNU, etc.))

Physical Connectivity:

- The only physical connectivity requirement is a supported I/O port on the device (USB, parallel/IEEE-1284, etc.).

CPU Bandwidth:

- CPU Bandwidth needs depend on the type of printing support required as well as what other operations are occurring in the system at print time.
- **For graphics support:**
  - A 100 MHz Pentium class processor (necessary for development but not for deployment)
- **For simple, ASCII-only text support:**
  - A low-end processor.

Available Memory:

- Code & static tables (50 - 120K).
- Run-time buffer space (25 - 60K, possibly 120K for high-res 6 color).

Code size and run-time RAM usage depend on several variables including the compiler, the printer types selected, the level of print functionality required (such as the optional ASCII text support or image scaling module), and the type and number of DeskJet models supported. As with anything else, compiled code size is also dependent on the targeted processor type.

9.5.4 Building Your Driver

The following is a skeletal makefile to compile the core source files. Auxiliary files containing components such as the code to implement the **SystemServices** (p. 33) object as well as system-specific compiler options and linking commands, must be added to your makefile.

Note:

There are several different versions of make, not all of which support the same code. This makefile may not work on all platforms. An excellent source of information on Makefiles is, Managing Projects with make, by Andrew Oram & Steve Talbott.

In order to maintain simplicity, (and because developers should not change any of the core files,) the only dependencies shown are of the object (.o) files upon the source (.cpp) files. Note however, that with some versions of make, a separate rule is required for each of these individual object (.o) files.

The skeletal makefile also lists the build symbols that define basic build parameters, including what printer classes will be included in the build. These classes do not match specific DeskJet printer models, but rather group all models into equivalence classes representing the types of ink cartridges or other features requiring separate code paths.

The purpose of providing these compiler switches is to allow separate drivers of significantly smaller size to be built - presumably to be downloaded into devices after the printer model connected to the user's device has been identified, (either automatically or with user input). If code size is not an issue, simply include all of the printer-model compiler switches.

Symbols also exist to provide support for the various font families when device-text is desired. If no font families are selected, all device-text support is removed from the build. Symbols also exist to correctly compile for little_endian vs. big_-endian platforms.

```
# flags for build:
    # APDK_CAPTURE for generating scripts only
    # APDK_LITTLE_ENDIAN if platform uses little-endian byte-order
    # APDK_NO_FONTS to exclude device-text support
    # printer-model selection:
    # DeskJet600c APDK_DJ600
    # Series 600 including 660,670,680 APDK_DJ6xx
    # Series 600 including 610,640,690 APDK_DJ6xxPhoto
    # Series 800 APDK_DJ8xx
    # Series 900 including 970, 950, 930 APDK_DJ9xx
    # Series 900 including 990 APDK_DJ9xxVIP
    # Apollo P-2100 series APDK_APOLLO2100

    # DEFINE FLAGS HERE
    # example: production build for all series 600 printers,
    # with text support, for Intel platform
    # FLAGS = -DAPDK_LITTLE_ENDIAN -DAPDK_DJ600 -DAPDK_DJ6xx -DAPDK_DJ6xxPhoto

CC = g++ -c $(FLAGS)

.cpp.o:
            echo "compiling $(<F)"
            $(CC) $(<F) -o $(*F).o
```

```
OBJS = dj9xxvip.o apollo21xx.o breaks_open.o dj8xx.o capture.o cgtimes.o courier.o \
       compression.o context.o create_so.o creator.o \
       dj600.o dj630.o dj660.o dj690.o dj6xx.o dj8x5.o dj600_maps.o dj660_maps.o \
       dj690_maps.o dj895_maps.o dj895_maps2.o dj970_maps.o dj970_maps2.o \
       filterhpa.o font.o globals.o header.o htfed_open.o imager.o imager_open.o \
       interp_open.o job.o ltrgothic.o printer.o registry.o ressynth.o \
       scaler_open.o script.o systemservices.o textmanager.o \
       textsupport.o translator.o univers.o version.o versioncode.o
```

9.5.5 Creating Custom Printer Drivers With The APDK

While it is expected that the developer will have to write some O/S or device-specific code in order to implement a printer driver on to their host system, the APDK has been structured in such a way that new code incorporated by the developer should be isolated from the core code. As such, the required changes to the core code should be limited to the object or module responsible for creating an O/S (Operating System) and hardware abstraction layer for the rest of the driver. This is called the **SystemServices** (p. 33) object.

If the developer discovers a defect in the device-independent portion of the APDK code, they are encouraged to fix the problem in their own driver.

Upon completion of the printer driver, Hewlett-Packard highly recommends that developers test their printer drivers using the test documents included with the APDK, or accessible via the SPP website.

HP no longer provides any support for developing or debugging APDK based drivers.

9.6 Release Notes

To view the full documentation in html use a web browser to open *index.html*.

Version 04.11.000 (p. 47) **Version 04.10.000** (p. 47) **Version 04.09.000** (p. 48) **Version 04.08.000** (p. 49) **Version 04.07.000** (p. 50) **Version 04.06.000** (p. 50) **Version 04.05.000** (p. 51) **Version 04.04.000** (p. 51) **Version 04.03.000** (p. 52) **Version 04.02.000** (p. 53) **Version 04.01.000** (p. 53) **Version 04.00.000** (p. 54) **Version 03.09.001** (p. 55) **Version 03.09.000** (p. 56) **Version 03.08.000** (p. 57) **Version 03.07.000** (p. 58) **Version 03.06.000** (p. 59) **Version 03.04.00A** (p. 60) **Version 03.03.00A** (p. 61) **Version 03.02.00** (p. 62) **Version 03.00.01** (p. 63) **Version 3.0** (p. 65) **Version 2.4** (p. 67)

9.6.1 Version 04.11.000

Changes Done

- Modified the accompanying documents to remove references of Support for developing and Debugging APDK based drivers
- Modified APDK Licence to remove "HP Confidential" tag

9.6.2 Version 04.10.000

New Model Support

- Deskjet 2050 J510
- Deskjet 1050 J410
- Officejet 4400 K410
- Officejet 4500 All-in-One Printer - K710
- Officejet 4500 G510
- Photosmart D110
- Photosmart B010
- Photosmart B110

New Features

- Brochure MediaType (enum value 8) option added. Also 5 new print modes added, namely Brochure Normal and Best, Premium Normal and Best, Plain Best.
- New **PrintContext** (p. 22) method added to set Media Subtype void **PrintContext::SetMediaSubtype** (p. 31) (int iMediaSubtype)
- Options to set Overspray values added to **PrintContext** (p. 22) method DRIVER_ERROR **PrintContext::SetPrinterHint** (p. 32) (PRINTER_HINT eHint, int iValue)
- New **PrintContext** (p. 22) method added to set mechanical firmware offset void **PrintContext::SetMechOffset** (p. 25) (int iMechOffset)

Bug Fixes

- FLSA paper size was using USLegal Paper size(value 3) for PageSize command. Corrected to use FLSA paper size(value 10) for PageSize command.

Known Issues

9.6.3 Version 04.09.000

New Model Support

- Deskjet D1600 series
- Deskjet D5500 series

- Deskjet F2400 series
- Deskjet F4500 series
- Deskjet Ink Advant K209a-z
- Deskjet Ink Advant K109a-z
- Officejet Pro K5600 series
- Officejet Pro L8200, L8300 L8400 series
- Officejet 6500 E709a, E709n Series
- Officejet 7000 E809a Series
- Photosmart C309a series
- Photosmart Plus B209a-m
- Photosmart C4700 series
- Photosmart Prem-Web C309n-s
- Photosmart Premium C309g-m
- Photosmart B109a-m
- Photosmart Wireless B109n-z
- Photosmart A640
- LaserJet P3011/P3015

New Features

- No new features were added for this release.

Bug fixes

- Replaced strcpy and sprintf with strncpy and snprintf
- Fixed various compiler warnings to help Java wrapper code
- Fix for memory de-allocation error in scaler.cpp
- Fix for thin black line down the page due to uninitialized buffer

Known Issues

9.6.4 Version 04.08.000

New Model Support

- Deskjet D2600 series
- Deskjet D4300 series
- Deskjet F4400 series
- Deskjet F4213 series
- Officejet 6000 E 609 series
- Officejet H470 series
- Officejet J4500 series
- Officejet J4600 series

- Photosmart C4600 series
- Photosmart D5400 series
- Laserjet M1522 MFP Series
- LaserJet P1005, P1006, P1007, P1008
- LaserJet P1505
- LaserJet P2035, P2055
- Color LaserJet 1600
- Color LaserJet 2600
- Color LaserJet CM1312 MFP
- Color LaserJet CP2025
- Color LaserJet CM2320 MFP
- Color LaserJet P3525

New Features

- Added page pre-pick (Speed Mech) support for DJ9xxVIP and derived class of printers.

Bug fixes

- Fixed a photo printing issue with some officejet printers. **Known Issues**

9.6.5 Version 04.07.000

New Model Support

- Deskjet D730
- Deskjet F2200 series
- Deskjet F735
- Deskjet F4210 series
- Deskjet F4213 series
- Photosmart A630 series
- Photosmart B8500 series
- Photosmart C4500 series
- Photosmart C5300 series
- Photosmart C5500 series
- Photosmart C6300 series
- Photosmart D7500 series
- Photosmart Pro B8800 series

New Features

Bug fixes

Known Issues

9.6.6 Version 04.06.000

New Model Support

- Deskjet D2500 series
- Deskjet F4200 series
- Officejet H470 Series
- Officejet J4500 Series
- Officejet J6400 Series
- Officejet Pro K8600 series
- Photosmart A530 series
- Photosmart C4340 series
- Photosmart C4400 series
- Photosmart D5400 series
- Photosmart D7200 series
- Photosmart D7400 series
- Photosmart Pro B8800 series

New Features

Bug fixes

Known Issues

9.6.7 Version 04.05.000

New Model Support

- Deskjet 6988
- Deskjet 6988dt
- Officejet J3600 Series
- Officejet J5500 Series
- Officejet K7100 Series
- Officejet Pro K8600 series
- Photosmart A320 series
- Photosmart A440 series
- Photosmart A520 series
- Photosmart A620 series
- Photosmart A820 series
- Photosmart C4380 series
- Photosmart C5200 series
- Photosmart C6200 series
- Photosmart C7200 series
- Photosmart C8100 series
- Photosmart D5300 series
- Photosmart D7200 series
- Photosmart D7400 series
- Color LaserJet CM4730 MFP
- Color LaserJet CP3505
- LaserJet P2010

New Features

Bug fixes

Certain Photosmart printers generated a printing error when printing on custom papersize. This has now been fixed.

Known Issues

9.6.8 Version 04.04.000

New Model Support

- Officejet J5700
- Officejet Pro K5300
- Officejet Pro K5600
- Officejet Pro L7300
- Officejet Pro L7500
- Officejet Pro L7600
- Officejet Pro L7700
- Photosmart C4200 series

New Features

Printing on to CD/DVD media is now supported on Photosmart-D5100 series printers. Set **Media Source** to **14** and **Media Type** to **MEDIA_CDDVD** (**7**). Two media sizes, **CDDVD_80** and **CDDVD_120** are supported, which are respectively 3 inch and 5 inch media. Note that it is still application's responsibility to clip the printable data to fit between the outer and inner circles of the media.

Bug fixes

Known Issues

9.6.9 Version 04.03.000

New Model Support

- Photosmart A430
- Photosmart A510
- Photosmart A610
- Photosmart A710
- Photosmart C3100 series
- Photosmart C7300 series
- Photosmart D5100 series
- Photosmart D6100 series
- Photosmart D7100 series
- LaserJet P2015
- LaserJet P3003

- LaserJet P3004
- LaserJet P3005
- LaserJet M3027 MFP
- LaserJet M3035 MFP
- LaserJet M5025 MFP
- LaserJet M5035 MFP
- LaserJet M4345 MFP
- Color LaserJet CM1015
- Color LaserJet CM1017
- Color LaserJet CP4005

New Features

Printing on to CD/DVD media is now supported on Photosmart-D5100 series printers. Set **Media Source** to **14** and **Media Type** to **MEDIA_CDDVD** (**7**). Two media sizes, **CDDVD_80** and **CDDVD_120** are supported, which are respectively 3 inch and 5 inch media. Note that it is still application's responsibility to clip the printable data to fit between the outer and inner circles of the media.

Bug fixes

Known Issues

9.6.10 Version 04.02.000

New Model Support

- Deskjet 6940
- Deskjet 6980
- Deskjet F300
- Officejet 4300
- OfficeJet 6300
- Color LaserJet 2605
- Color LaserJet 3050
- Color LaserJet 3060
- Color LaserJet 3062
- Color LaserJet 3390
- Color LaserJet 3392
- LaserJet 5200

Bug fixes

- Fixed endian related bug in LJJetready
- Moved PSC 2100, 2150 and 2170 from DJGenericVIP class to DJ9xx-VIP class

Known Issues

9.6.11 Version 04.01.000

New Printer Family Support

In this release, three new printer family support was added, namely, DJ55xx, OJProKx50 and PSP470.

New Model Support

- Deskjet 460
- Deskjet 5440
- Deskjet 5940
- Deskjet 6600
- Deskjet 9800
- Photosmart 330
- Photosmart 380
- Photosmart 420
- Photosmart 470
- Photosmart 2570
- Photosmart 3100
- Photosmart 3200
- Photosmart 3300
- Photosmart 7800
- Photosmart 8000
- Photosmart 8200
- Photosmart 8750
- Officejet Pro K550
- OfficeJet Pro K850
- Business Inkjet 2800
- Color LaserJet 2800
- Color LaserJet 3000
- Color LaserJet 3600
- Color LaserJet 3800
- Color LaserJet 4610
- Color LaserJet 4700
- Color LaserJet 4730 MFP
- LaserJet 4240
- LaserJet 9040

New Source Files

- dj55xx.h
- ojprokx50.h
- psp470.h

Enhancements

- Added support for 5x7 media
- Added new printmodes in DJGenericVIP class

Bug fixes

- Fixed margin error for some deskjets

– Moved Deskjet 450 to DJ9xxVIP class
– Moved Deskjet 55xx to the new class DJ55xx
– Fixed custom papersize bug in LJJetready and FastRaster printers
– Fixed a pen parsing bug in DJ9xxVIP

Known Issues

9.6.12 Version 04.00.000

New Printer Family Support

In this release, two new printer family support was added, namely, LJJetReady and LJFastRaster. LJJetReady requires JPEG library. The source code for JPEG library is not included in this distribution. The developers must download JPEG library source from publicly available sources if their system does not already have it. Please refer to README_LIBJPG for licensing issues.

New Model Support

– Deskjet 3528
– Deskjet 3535
– Deskjet 1280
– psc 1100
– psc 1200
– Officejet 4100
– Officejet 4105
– Officejet 4200
– Business Inkjet 1000
– Business Inkjet 1100
– Color LaserJet 3500
– Color LaserJet 3550
– LaserJet 1010
– LaserJet 1012
– LaserJet 1015

New Source Files

– ljjetready.cpp & ljjetready.h
– ljfastraster.cpp & ljfastraster.h

Enhancements

Bug fixes

Known Issues

9.6.13 Version 03.09.001

New Model Support

- Deskjet 6540
- Deskjet 6520
- Deskjet 5740
- Deskjet 6840
- Deskjet 3840
- Deskjet 3740
- Photosmart 7400
- Photosmart 8100
- Photosmart 8400
- Photosmart 320
- Photosmart 370
- Officejet 7200
- Officejet 7300
- Officejet 7400
- Photosmart 2600
- Photosmart 2700
- psc 1600
- psc 2350
- Officejet 6200
- Business Inkjet 1200
- Color LaserJet 2550
- Color LaserJet 5550
- Color LaserJet 9500 MFP
- LaserJet 9050
- LaserJet 9050 MFP
- LaserJet 9040 MFP
- LaserJet 1320
- LaserJet 1160
- LaserJet 4250
- LaserJet 4350
- LaserJet 2410
- LaserJet 2420
- LaserJet 2430

New Source Files

Enhancements

Bug fixes

Known Issues

Certain printers selected through DJ9xxVIP and DJGenericVIP class, typically Business Inkjets, always output in Palin Paper, Normal Print Mode. This happens with only those printers that do not have Media Sensing device in the printer.

9.6.14 Version 03.09.000

New Model Support

- Deskjet 6540
- Deskjet 6520
- Deskjet 5740
- Deskjet 6840
- Photosmart 7400
- Photosmart 8100
- Photosmart 8400
- Photosmart 320
- Photosmart 370
- Officejet 7200
- Officejet 7300
- Officejet 7400
- Photosmart 2600
- Photosmart 2700
- psc 1600
- psc 2350
- Officejet 6200
- Business Inkjet 1200
- Color LaserJet 2550
- Color LaserJet 5550
- Color LaserJet 9500 MFP
- LaserJet 9050
- LaserJet 9050 MFP
- LaserJet 9040 MFP
- LaserJet 1320
- LaserJet 1160
- LaserJet 4250
- LaserJet 4350
- LaserJet 2410
- LaserJet 2420
- LaserJet 2430

New Source Files

Enhancements

Bug fixes

Known Issues

Certain printers selected through DJ9xxVIP and DJGenericVIP class, typically Business Inkjets, always output in Palin Paper, Normal Print Mode. This happens with only those printers that do not have Media Sensing device in the printer.

9.6.15 Version 03.08.000

New Model Support

- psc 1310
- OfficeJet 4200
- OfficeJet 9100
- Business InkJet 2300

New Source Files

Enhancements

- KRGB Support for Improved Black Text Quality

The current APDK accepts RGB raster data as its input and generates printer ready data based on the printer format. The problem with RGB only data path is that black only objects in a document, such as balck text or line drawing objects, may get fuzzy edges, the color may not be true black, and the performace will suffer due to the extra colormatching and halftoning process. With the addition of the 1 bit K-plane data path feature, the host were provided with the option to send black only objects in an one bit per pixel format. This will address both the quality and the performance issues.

Bug fixes

- In scaler.cpp, pOutputBuffer is never initialized. Initialized it to all 0xff (white).

Known Issues

Certain printers selected through DJ9xxVIP and DJGenericVIP class, typically Business Inkjets, always output in Palin Paper, Normal Print Mode. This happens with only those printers that do not have Media Sensing device in the printer.

9.6.16 Version 03.07.000

New Model Support

- DeskJet 5650
- DeskJet 5670
- DeskJet 5150
- DeskJet 5160
- DeskJet 5158
- DeskJet 5850
- DeskJet 3650
- DeskJet 3550
- DeskJet 3520
- DeskJet 3668

- DeskJet 3658
- DeskJet 3558
- DeskJet 3528
- DeskJet 9300
- deskJet 9600
- PSC 2300
- PSC 2400
- PSC 2500
- PSC 2170
- psc 1300
- OfficeJet 5500
- OfficeJet 7100
- OfficeJet 4100
- OfficeJet 6100
- OfficeJet 6150
- LaserJet 2300
- LaserJet 1300
- Business InkJet 1100
- Photosmart 240
- Photosmart 140
- Photosmart 7960
- Photosmart 7760
- Photosmart 7660
- Photosmart 7260
- Photosmart 7268
- PostScript Printers

New Source Files

- dj3600.cpp
- dj3600.h
- dj3600_cmap.cpp
- pscript.cpp
- pscript.h

Enhancements

- Printer Selction Interface

You can now select the printer by model name or family name in addition to printer type.

Bug fixes

- No bugs reported

Known Issues

Certain printers selected through DJ9xxVIP and DJGenericVIP class, typically Business Inkjets, always output in Palin Paper, Normal Print Mode. This happens with only those printers that do not have Media Sensing device in the printer.

9.6.17 Version 03.06.000

New Model Support

- DeskJet 3320
- DeskJet 3400
- PSC 1100
- PSC 1200
- LaserJet 4200
- LaserJet 4300
- Business InkJet 3000
- OfficeJet 5100

New Source Files

- dj3320.cpp
- dj3320.h
- dj3320_cmap.cpp
- printerfactory.cpp
- printerfactory.h
- printerproxy.cpp
- printerproxy.h

Enhancements

- Printer Selction Interface

You can now select the printer by model name or family name in addition to printer type.

Bug fixes

- No bugs reported

Known Issues

Certain printers selected through DJ9xxVIP and DJGenericVIP class, typically Business Inkjets, always output in Palin Paper, Normal Print Mode. This happens with only those printers that do not have Media Sensing device in the printer.

9.6.18 Version 03.04.00A

New Model Support

- Deskjet 6122
- DeskJet 6127
- DeskJet 3816
- DeskJet 450
- PhotoSmart 230
- PhotoSmart 7550
- Color LaserJet 2500

– Color LaserJet 4600
– Color LaserJet 4550
– Color LaserJet 5500

New Source Files

– ljcolor.cpp
– ljcolor.h

Enhancements

– Media Tray Selection

You can now select the media source on those printers that have multiple bins. See SetMediaSource API.

Bug fixes

– No bugs reported

Known Issues

Certain printers selected through DJ9xxVIP and DJGenericVIP class, typically Business Inkjets, always output in Palin Paper, Normal Print Mode. This happens with only those printers that do not have Media Sensing device in the printer.

9.6.19 Version 03.03.00A

New Model Support

– Deskjet 3820
– DeskJet 5550
– DeskJet 5551
– PhotoSmart 130
– PhotoSmart 7150
– PhotoSmart 7350
– LaserJet 1200
– LaserJet 2200
– LaserJet 4100

New Source Files

– ljmono.cpp
– ljmono.h

Enhancements

– Version number standard now documented. All version numbers for the APDK are now based on 6 digits and 1 letter. The format is mm.nn.ffx where:

* mm - is the major version number. This changes when significant API changes are made or major class/object code change.
* nn - is the minor release number. This changes when new printer support is added, internal or compatible API changes are made.
* ff - fix level. This changes when bugs are fixed or improvements are made. Often these versions are internal and fixes are rolled into the next minor release.
* x - back-patch version. Will be 'A' for a release. Used when we need to patch an older version of the APDK for a specific use or reason. The letter is incremented for every back-patch of a version we create.

Bug fixes

- No bugs reported

Known Issues

Black vertical band on paper, or ink sprayed on platen when ALL of the following conditions are true:

- VIP class printer (APDK_DJ9xxVIP)
- APDK built with device font support (i.e. APDK_CGTIMES, APDK_COURIER, APDK_UNIVERS, or APDK_LTRGOTHIC)
- Letter-size job never sent to printer
- Media size set to A6 or narrower

If all of the above are true, but there is letter-size media in the printer then a black band will be printed on down the right side of the page. If the media in the printer is same as requested size then black ink will be sprayed on the platen. Once a job with a media size of A4 or larger is sent to the printer then this behavior disappears.

Note:

The use of device text is still discouraged and this issue only exists when device text is on. This cannot be resolved in the APDK code and will remain a known issue for VIP printers when using device text support.

9.6.20 Version 03.02.00

New Model Support

- Business Inkjet 2200
- Business Inkjet 2230
- Business Inkjet 2250
- Business Inkjet 2280

New Source Files

- djgenericvip.cpp

– djgenericvip.h

Enhancements

– Licence has changed - now all APDK code is under BSD open source license. Please read the new license (at the top of each source file).
– Speed improvements in half toning and compression
– APDK_HIGH_RES_MODES build option added.

  This will include code for 1200 or 2400 DPI for some printers. Do not use this option unless you have input data near 1200 DPI. This is a CPU and I/O intensive option.
– New APIs
  * BOOL **PrintContext::PhotoTrayPresent**(BOOL bQueryPrinter) (p. 23);
    Returns TRUE if a photo tray is present in printer.
  * PHOTOTRAY_STATE **PrintContext::PhotoTrayEngaged** (p. 23) (BOOL bQueryPrinter);
    Returns current state of phototray, one of UNKNOWN, DISENGAGED or ENGAGED.
– The APDK now avoids fractional scaling if the output is within .02 inches of a multiple of the input. Fractional scaling is much more complex and CPU intensive then pixel replicaion, which can be performed if the output is a multiple of the input.

Bug fixes

– Bug fixes to improve reliability and functionality.
– Fixed DJ8x5 print problems when grey CMY requested with only a black pen.
– Missing "dj8x5.h" file restored.

Known Issues

None

9.6.21 Version 03.00.01

Enhancements

– We continue to improve the documentation delivered with the APDK. We are interested to know if you find the documentation for version 3.0 and later to be an improvement or not. For any comments, to report documentation errors, or make suggestions on needed documentation please send e-mail to: `APDK Support`.
– APDK_VIP_COLORFILTERING directive added.

  By default the color filtering code for VIP printers is now eliminated from the build. To turn color filtering on define APDK_VIP_COLORFILTERING. Non VIP printers do not use color filtering

(defining APDK_VIP_COLORFILTERING will not include the code unless a VIP printer is also defined). Eliminating color filtering is an advantage for systems with low CPU resources but enough I/O bandwidth to drive the printer. For systems with more CPU cycles available and/or lower I/O bandwidth then turn on color filtering by defining APDK_VIP_COLORFILTERING if you are enabling a VIP printer. Previously, color filtering code was included, even if it could not be used.

– APDK_NO_NAMESPACE directive added.

  Because of continuing name conflicts with code integration of current projects and products all classes, types, variables, etc... have been placed in the '**apdk** (p. 18)' name space. This will require that the developer include a line "using namespace APDK_NAMESPACE;" after the inclusion of "hpprintapi.h". Alternately, all references to APDK classes, types, variables, etc... can be prefixed with APDK_NAMESPACE:: (for example APDK_NAMESPACE::PrintContext or **APDK_NAMESPACE::DRIVER_ERROR** (p. 5)). Lastly, if your compiler does not support name spaces or they are just not working for you then define APDK_NO_NAMESPACE and the APDK will be built without name spaces (i.e. in the global namespace). Note: APDK_NAMESPACE is a macro that translates to '**apdk** (p. 18)' unless APDK_NO_NAMESPACE is defined, in which case it translates to nothing.

– Many *#define*s have been converted to *const int* or *enum*. This gives better type checking and allows these items to be placed in the **apdk** (p. 18) name space. Many of the *#define*s that now exist are macros or support the C interface to the APDK. There will be continuing efforts to reduce *#define*s and improve type checking, readability, and robustness. Expect to see these in future versions.

– Additional paper sizes are now supported. See **PAPER_SIZE** (p. 7) for details on supported sizes.

  Note:

  Full Bleed paper sizes will be removed in the future in favor of defining just standard paper sizes and allowing the developer to select a full bleed option for any paper size selected.

Bug fixes

– GREY_CMY now works for all printers. Specifying GREY_CMY as a the COLORMODE in SelectPrintMode will use color ink to produce grey scale output. Although this uses color ink instead of black the output is much nicer. Specifying GREY_K uses the black pen only but appears grainier.

  Note:

  If the black pen is available on a VIP family printer then GREY_CMY will fortify the CMY grey scale with black ink also.

- **SystemServices::FromDevice** (p. 37)(...):

  The signature for FromDevice did not match the signature for ToDevice and was incorrect. It was necessary to change the parameter types in FromDevice. This method is a pure virtual function and as such requires the same changes in any derived classes from **SystemServices** (p. 33). See documentation on **SystemServices::FromDevice** (p. 37) for more information.
- **PrintContext::GetPrintModeSettings**() (p. 29) change:

  A prototype API (GetPrintModeSettings) managed to make it into version 3.0. Unfortunately, this API had memory leaks and required the code integrator to free memory allocated by the APDK. Set the new GetPrintModeSettings API for the interface changes.
- Paper Sizes:
  - Both Hagaki and Oufuku paper size coordinates were reversed (i.e. x & y values swapped).
  - enum for Oufuku was misspelled as OFUKU, new entry created for OUFUKU.
  - A4_FULL_BLEED should have really been PHOTO_FULL_BLEED. Both enums exist now (PHOTO_FULL_BLEED is the proper name).
  - SetPaperSize nows allows setting paper size to any (supported) size smaller than mandatory size, not just the mandatory size.
- There were several memory leaks in the new SelectPrintMode API that have been fixed.
- **PrintContext** (p. 22) APIs would attempt to access NULL pointers under certain conditions.

Known Issues

- If device text is enabled and any grey scale mode is selected then the text will still appear in color (i.e. not snapped to black) if a color was requested. In the future all text will be snapped to black if a grey scale mode is requested.
- Under some conditions generating a debug script using APDK_CAPTURE fails. Everything appears to work but the generated script file is corrupt.

9.6.22 Version 3.0

New Model Support

- Deskjet 656
- Deskjet 825 & 845
- Deskjet 920
- Deskjet 940 & 948
- Deskjet 995
- Deskjet 1125C

- Deskjet 1220C
- Photosmart 100
- Photosmart 1115
- Photosmart 1215
- Photosmart 1315
- CP 1160
- CP 1700
- Apollo P2500
- Apollo P2600

Source Files

New source files that have been added are:

- apollo21xx.cpp & apollo21xx.h
- apollo2560.cpp & apollo2560.h
- apollo2xxx.cpp & apollo2xxx.h
- dj8x5.cpp & dj8x5.h
- dj8xx.cpp & dj8xx.h
- phobos_cmaps.cpp
- pmselect.cpp & pmselect.h
- psp100.cpp & psp100.h

Source files that have been removed:

- aladdin.cpp & aladdin.h
- ap2100.cpp & ap2100.h
- broadway.cpp & broadway.h
- dj895.cpp & dj895.h

Developer's Guide

Significant changes to the developer's guide. Formatting and indexing changes to the guide, including improved API documentation. Open index.html in the DevGuide directory to open the html version of the documentation. A printable version of the guide is available in devguide.pdf in the same directory.

New Features

New and deprecated APIs for version 3.0

New print mode selection:

SelectPrintMode(int) has been deprecated - new code should not use it. Existing code should migrate to new SelectPrintMode. This method still exists for backward compatibility but will be removed in a future version of the APDK.

New **PrintContext** (p. 22) method SelectPrintMode(QUALITY_MODE, MEDIATYPE, COLORMODE, BOOL); Please see API documentation for details.

GetModeName() has been deprecated. This method returned US English ASCII strings. It is now up to the developer to proved strings to be used in the user interface. GetModeName now always returns an empty string and will be removed in a future version of the APDK.

Use PRINTMODE_VALUES *GetPrintModeSetting() to determine the quality, mediatype, colormode, and device text settings for the current print mode.

DRIVER_ERROR SetPenSet(PEN_TYPE ePen) allows the caller to specify the pen set in the printer in uni-di mode.

Known Defects

Selecting a COLORMODE of GREY_CMY will still print full color images on some printers. Selecting GREY_K will properly use the black pen to print grey scale images. Selecting GREY_CMY should (and will in future versions) print a grey scale image using the color pen. Although this uses color ink the image is much nicer.

If device text is enabled and any grey scale mode is selected then the text will still appear in color (i.e. not snapped to black) if a color was requested. In the future all text will be snapped to black if a grey scale mode is requested.

9.6.23 Version 2.4

Source Files

Downloaded source.zip file contains the following source code files:

Note:
all filenames are now lower-case; previous releases used mixed-case filenames.

- aladdin.cpp
- aladdin.h
- ap2100.cpp
- ap2100.h
- breaks_open.cpp
- broadway.cpp
- broadway.h
- capture.cpp
- cgtimes.cpp
- compression.cpp
- compression.h

- context.cpp
- courier.cpp
- create_so.cpp
- creator.cpp
- debug.h
- dj350.cpp
- dj350.h
- dj600.cpp
- dj600.h
- dj600_maps.cpp
- dj630.cpp
- dj630.h
- dj660.cpp
- dj660.h
- dj660_maps.cpp
- dj690.cpp
- dj690.h
- dj690_maps.cpp
- dj6xx.cpp
- dj6xx.h
- dj895.cpp
- dj895.h
- dj895_maps.cpp
- dj895_maps2.cpp
- dj970_maps.cpp
- dj970_maps2.cpp
- ernieplatform.h
- filterhpa.cpp
- filterhpa.h
- font.cpp
- font.h
- globals.cpp
- global_types.h
- harness.h
- header.cpp
- header.h
- hpprintapi.h
- hpprint_c_api.cpp
- hpprint_c_api.h
- hptypes.h
- htfed_open.cpp
- imager.cpp
- imager.h
- imager_open.cpp
- imaging.h
- internal.h
- interp_data_50.h
- interp_open.cpp
- io_defs.h
- io_utils.cpp

- job.cpp
- ltrgothic.cpp
- models.cpp
- models.h
- platform.h.readme
- printer.cpp
- printer.h
- registry.cpp
- resources.h
- ressynth.cpp
- rgbtocmyk.cpp
- scaler_open.cpp
- scaler_open.h
- scaler_prop.h
- script.cpp
- script.h
- systemservices.cpp
- textmanager.cpp
- textsupport.cpp
- translator.cpp
- univers.cpp
- version.cpp
- versioncode.cpp

Documentation

Documentation has been included in an additional zip file (documentation.zip). Be sure to download the documentation and read the intro.htm for a documentation and source code overview. The documentation.zip file contains the following html documents:

- intro.htm
- gettingstarted.htm
- api.htm
- bestpractices.htm
- debugging.htm
- devicebasedfont.htm
- faqs.htm
- glossary.htm
- hostsysteminterface.htm
- lowlevel_io.htm
- printing.htm
- samplecode.htm
- testing.htm

New Model Support

- DeskJet980
- DeskJet960
- DeskJet350 (Requires new preprocessor symbol APDK_DJ350 added to your makefile)
- Photosmart 1000 / 1100
- Photosmart 1215 /1218 (Currently the photo tray is not supported on these models)

New Features

- A6 Paper size support for all printers that support A6

Debug Information

Remember to edit the string DeveloperString in version.cpp, so that when debug information is generated it will identify your platform.

Limitations on Apollo2100/2150

There is a problem in identifying the pen configuration for these models. As a result the APDK has implemented a workaround: we assume the color pen alone is present. The black pen will not be used, either for color OR grayscale modes. Selecting grayscale as the print-mode will work only if the color pen is present.

9.7 Best Practices For Optimal Printing

It is easy to make the mistake of assuming that the image on the screen will automatically transfer to the printed page, printing an identical image as to what is on the screen. It does not.. This screen shot' approach will not achieve the look and feel that a user expects when holding printed output in thier hand, and comparing that same information to what is on the TV screen. The main differences between a video display (TV/monitor) and a printout of that display are in image resolution levels and formatting for correct text size and color.

9.7.1 Image/Text Resolution

A higher resolution image (more pixels per inch) means a higher quality image. Therefore it is not surprising that when image data is compressed (such as compressing a multi-megabyte bitmap image down to a 125k

JPEG image,) some amount of data (i.e. quality) loss occurs in the process. In other words, higher compression leads to greater quality degradation. On the opposite end of the spectrum, low resolution compressed images may look fine on a video screen at 72 dpi, but will not look very good after they have been scaled back up to the 300 or 600 dpi requirements of the printer. The images will be blocky - a curved line for instance will not be smooth but instead have a stair-step appearance. Text appears to suffer more than images because smaller objects rely more on the precision of high resolution, and while a low resolution image may simply have poor print quality, low- resolution text may become unreadable.

Best Practice: Use the highest quality images available when printing. The ability to deliver this depends on the capabilities of the host system, but the same image used by the display does not have to be the same one used for printing. For example, the server for an internet device may compress image data to improve download speed. The device itself may receive the image data and down-sample' the image even further, essentially throwing away data to fit the 72 dpi screen. In this case, it would be desirable to re-request the uncompressed version of this image when formatting for the higher resolution printed page. Go back to the original image data and ask that the page be re-formatted for printing, rather than simply passing off the 72 dpi screen page to the printer driver.

For text, if the device has a font-rendering engine, the text for the printed page should be re-rendered through the font engine at the higher resolution when formatting the page for printing. (Refer to **Device Fonts** (p. 72) for more information.) Alternately, if the device simply has a set of stored raster fonts optimized for screen resolution, the developer should make use of the printer's high-resolution built-in fonts. Text rendered at 72 dpi screen quality and then passed to the printer driver will produce the same poor quality as stored 72 dpi raster fonts. Your font engine is a powerful tool - take advantage of it!

9.7.2 Formatting and Layout for the Screen .vs. the Printer

Reading text on a television screen from across the room or even from an LCD screen attached to the phone is different from reading text on the printed page. Passing a page' of data optimized for the phone or television screen directly to the printer will produce a printed page which is WYSIWYG ("What You See Is What You Get"), but not what the user expects. The news article the user prints from their favorite on- line news service should not be 5 pages of BIG text if that same article would have taken one page in the local newspaper.

Best Practice: Normal text should be about 10 to 12 points in size and take advantage of the full paper size. A system with a 40 character LCD display may now have twice that width in which to format the information.

A TV/internet browser will now have much more text displayed on a page. As such, care must be taken to wrap text correctly around images. Actually, smaller text allows more precise placement and should wrap more easily around other objects. Be sure to take into account page and text colors. The printed page by default is white. But light colored text on a black background is common for TV based systems as it reduces noise and is generally easier to read. Laying down all the black ink on the printed page to reproduce this effect will certainly irritate the user who is waiting for all that ink to dry, and who must now go out and buy a new ink cartridge after printing relatively few print jobs. Take note of the change in background and text when choosing text color. White text will obviously not show up on a white background and even yellow text will be extremely difficult to read. Choose text colors that will contrast appropriately with the background. The system may make it an option to print backgrounds, as some web pages are designed such that the background image is an integral part of the page. If this option is chosen, we suggest the logic exists to not re-map the text color. (Mapping white text to black while printing a dark background would lead to the same problem of printing white text on the white page.)

Note:

If the ability to print page backgrounds is allowed, it should not be the default. Most backgrounds are extraneous to the information the user is printing.

9.7.3 Clipping to the Printed Page

A web page is usually formatted as one long page with text and graphics intermingled as appropriate. The printed page unfortunately has physical limits to its width and length. It is entirely possible and probable that while printing multi-page documents, text or images could be cut in half between pages.

Best Practice: Do the best you can. No easy set of rules exists to solve this problem in all scenarios. Be aware of how much space remains on the current page. The printable width and height of the page are available via the **PrintContext** (p. 22) object.

Here's a simple calculation to give you the input height (number of your rasters) that will fit on a printed page:

```
InputHeight = PrintHeight / (PrintWidth/InputWidth);
```

(PrintWidth/InputWidth) gives the scaling factor for the page. This value is then divided into the PrintHeight to supply the number of input rows. Tracking the number of rasters you've imaged to a given location will allow you to correspondingly track the amount of space remaining on the page.

Implement a "look-ahead" region so that text that would normally be clipped is moved to the next page. Images can be treated the same way,

though certainly some images may have to be clipped in order to maintain proper formatting. A very long image for instance, would require clipping, but smaller images may be nudged one way or the other to be kept intact. Certainly text is the easier and more critical of the objects to not have clipped.

9.8 Device Fonts

This document provides the information needed to understand the relationship between Device-based text and the Hewlett-Packard Appliance Printing Software Development Kit (APDK). Device-based text is simply font types that are built into the printer. The availability of built-in fonts depends on the printer model and print mode. (In some modes, no fonts are available.)

Rather than rendering text on the host system, text can be sent as a stream of ASCII code to be rendered by the printer firmware using one of its built-in fonts. Device-based fonts can often result in much sharper output depending on the host system facilities for rendering text at high resolutions and merging with graphics. Learn more about device-based fonts by reviewing the below topics.

9.8.1 Overview

Device-based fonts are accessed by an assortment of functions that allow page positioning of textual strings, and access to certain font treatments such as fixed vs. proportional spacing, text size, and bold, italic or underline.

The advantages of using device-based text depend on the font rendering capabilities of the host system. In other words, device-based fonts are more advantageous for host environments that do not have a font-rendering engine and instead rely on a set of pre-rendered bitmap fonts stored in memory. These pre-rendered fonts are typically stored at screen resolution (72 dpi,) and print relatively poorly. Use of the device-based fonts offers a dramatic increase in print quality.

Another advantage of using device text is speed - especially in cases where the print document is text-only. Compared to raster data, the printer driver has very little processing to perform on device text. Since device text is sent in ASCII format, the amount of data sent to the printer is much smaller and therefore can be sent more quickly. The printer is also able to store more text in ASCII format in its buffers and process the text more quickly.

The disadvantages of using device text are found in flexibility and formatting. Some calculations must be performed by the host system for page layout including scaling device text to the right size, and properly lining

up device text and image graphics data. In addition, only limited ranges of text sizes are available. Depending on the printer model, these are usually in the 6 to 14 point range, making ideal page formatting potentially a little more difficult. However, larger text can be achieved by sending it as part of the image data while the smaller text remains device-based. For readability, it is more important that smaller text be of higher quality.

9.8.2 Supported Device Fonts

Some Hewlett-Packard DeskJet printers support the following device fonts:

- Courier (Fixed Pitch/ Serif) - 6, 12, 24 pitch
- LetterGothic (Fixed Pitch/ Sans-Serif) - 6, 12, 24 pitch
- CGTimes (Proportional/ Serif) - 6, 8, 10, 12, 14 point
- Univers (Proportional/ Sans-Serif) - 6, 10, 12, point

Treatments available for these fonts:

- Bold
- Italic
- Underline

To enable the use of device fonts one or more of the following directives must be used in the build. Defining these includes the necessary code to support the device font. Without the directive the code is eliminated from the build to reduce the overall size of the driver.

- **APDK_COURIER** (p. 14)
- **APDK_LTRGOTHIC** (p. 14)
- **APDK_CGTIMES** (p. 14)
- **APDK_UNIVERS** (p. 14)

Using device-based fonts entails using some or all of the following API functions:

- PrintContext::EnumFont
- PrintContext::RealizeFont
- **Font::GetName** (p. 18)
- **Font::IsBoldAllowed** (p. 18)
- **Font::IsItalicAllowed** (p. 19)
- **Font::IsUnderlineAllowed** (p. 19)
- **Font::IsColorAllowed** (p. 19)
- **Font::IsProportional** (p. 19)
- **Font::HasSerif** (p. 19)
- **Font::GetPitch** (p. 20)
- **Font::GetTextExtent** (p. 18)
- **Job::TextOut** (p. 21)

9.8.3 Enumerating Available Fonts

A typical usage scenario might be to first find the fonts available (if any), by implementing an EnumFont while loop and querying the reference font for its properties. The pass-by reference iCurrIdx will be returned to 0 after a full cycle of the available fonts.

Font Example Note thePC' is a pointer to a previously instantiated **PrintContext** (p. 22) object. The following code sample looks for proportionally spaced font or a fixed pitch font that are sans- serif:

```
int PropSerifIdx = 0;
int FixedSerifIdx = 0;
int iCurrIdx = 0;
ReferenceFont* currFont = NULL;
while( currFont = thePC->EnumFont( iCurrIdx)) != NULL )
{
    if ( (currFont->IsProportional() == TRUE) &&
        (currFont->HasSerif() == FALSE) )
        {
            PropSerifIdx = iCurrIdx;
            break;
        }
        if ( (currFont->IsProportional() == FALSE) &&
        (currFont->HasSerif() == FALSE) )
    {
            FixedSerifIdx = iCurrIdx;
            break;
    }
} //while
```

9.8.4 Realizing the Desired Font

Once the type of font is found that has the desired attributes, its index value (in this example, PropSerifIdx), is used to realize' an actual font object as shown in the following code fragment. Refer to the RealizeFont API reference for the complete function.

```
Font* fntPropSerif = thePC->RealizeFont(PropSerifIdx, ... )
```

Multiple fonts may be instantiated at one time via this mechanism creating some flexibility in implementation; developers may choose to realize only a few font styles such as a proportional font and a fixed pitch font, and change treatments (i.e. bold on/off,) on the fly by manipulating public variables in the font class (fntPropSerif->bBold = TRUE).

Developers may also choose to realize font objects [font style & treatment] for each or several of the text objects on the page ahead of time. Creation of this font list or array would allow walking down the page and simply sending the correct font where applicable. This is more applicable when a set number of fonts/treatments are expected, as system resources may prohibit having too many font objects instantiated at one time. These implementation decisions are of course on a system by system basis.

9.8.5 Page Formatting and GetTextExtent

Proper page formatting requires the host to know how much space the text string takes up. This is necessary in order to find out if the text will fit on one line or if it will overlap a graphic object on the page. For this purpose, GetTextExtent will accept a text string and return its length and height, (the number of printer pixels from the baseline to the top of the tallest character of this string based on the font and size on a 300 dpi grid). Tracking this length and height is vital to correct placement of text on the page. Each TextOut call will require page placement coordinates.

9.8.6 TextOut

TextOut is the function for sending a device text string to the printed page. TextOut is conceptually quite simple, taking as parameters the text string, number of characters in the string, font, and x & y coordinates. The coordinates x & y specify the text currently being sent to the printer. The baseline is the invisible line upon which the line of text rests; the descender of a character (like y or g) hangs below this baseline. The x & y coordinates are of course critical to the correct layout of a page.

The *X* Coordinate

The x coordinate is used for text placement across the width of the page and is extremely relevant when text is being placed next to a graphic, or if multiple text strings are placed on the same line. Because text may have different attributes (such as a word in the middle of a line being Bold,) it may require several TextOut calls to compose this line.

Example: This is a sample line of text with **a Bold text string** in the middle.

In this example, the first part of the line is sent via TextOut at the left margin (x = 0). Next is a call to TextOut for the Bold text string for which the x coordinate is based off the length of the first text string. The final text string is again normal text and its x coordinate is based off the length of the two previous TextOut calls. The length of these strings is made available by first calling GetTextExtent with each of these strings, as described previously in GetTextExtent.

The *Y* Coordinate

The y coordinate is used for text placement along the height of the page. Two important factors exist for the Y placement of text; both relate to remembering that the Y coordinate is at the baseline (not the top) of a piece of text.

First, recall that GetTextExtent returned a height value. This is the number of printer pixels from the baseline to the top of the tallest character for the string, based on the font and size. For proper placement of consecutive lines of text, the Y coordinate used must allow for the height of the string and a few pixels of spacing (or leading) between the lines of text. If this calculation is not performed correctly, the consecutive lines of text will appear scrunched together or may overlap.

As a result of the Y coordinate (baseline) being at the bottom of the text string, it is important that the text be sent early enough for the printer to factor this in before printing the top portion of the text. The printer's memory buffer is capable of storing an entire page of text, so if the host's page formatting code allows all of the device text to be sent ahead of time, this may be a preferable option.

In other systems, the page formatter may only look at one band of data at a time in order to format a mixed device-text and raster-graphics print job. In such cases it is a common error to send the text "too late". Each row of raster data sent to the printer (SendRaster) will increment the printer mechanism in order to print the row. As mentioned, if the TextOut call is not made soon enough to compensate for the height of the text, the top portion of the text may be clipped and the printer will print only the lower section of text as determined by the current position of the mechanism on the page.

9.9 Host Interface

The Hewlett-Packard Appliance Printing Software Development Kit (APDK) enables system developers to produce a minimal, yet flexible, user-interactive printer driver. Developers are free to create a minimum number of user-interactive commands, or create additional commands by migrating driver error/status codes into Graphic User Interfaces (GUIs) and messages.

The below sections enumerate the use of user commands and interfaces within the printer driver.

9.9.1 User Interfaces & Guidelines

Drivers built from the APDK core code provide simple printing solutions that do not require extended responses from end users. Developers determine their own number and complexity of GUIs to be presented by their product to the user. The APDK does not implement any GUI.

There are a few basic concepts that are necessary to understand in order to instantiate the user interface for a print solution. The simplest concept is to have only one print button (or GUI) allowing the user to initiate a print job. Developers have the option of increasing printer/user interaction by

creating a second GUI for example, which the user could use to determine the CurrentMode or other print job options. These other options include:

- Print Area
- Margins
- Rendering and Formatting Options (e.g. eliminate solid backgrounds and invert font colors, send text only)
- Duplex Printing
- Output Tray Selection
- Printer Maintenance Functions (e.g. clean pen)

The following two lists are strongly recommended GUIs and optional GUIs. These lists are recommendations only, and should be modified to meet the needs of the host system and end user.

Strongly Recommended

- Error Dialogue - Errors that occur at the printing level are reflected to the user in an easily understandable manner.
- Printing Progress - The user visually sees the status of their print job.
- Printer-Model Selection (for Uni-di) - In cases where Bi-directional communication is not available, the user should be prompted to select their printer from a supported printer list.
- Driver Version - Allows the user to find out what Hewlett-Packard APDK core code version is being used.
- Printer Maintenance Functionality - Some functionality for printer maintenance should be incorporated. Future maintenance functionality will continue to be added to future APDK releases.

Optional

- Printmode Selection (including dialogue for mode mismatch) - Allows users to choose print modes such as print quality, grayscale, etc.
- **Font** (p. 18) Selection & Formatting (e.g. for e-mail) - Developers may want to give end users a minimal selection of fonts, or the ability to eliminate backgrounds or invert font colors when printing.
- Models Supported In Driver - Allows developers to show users which DeskJet models are supported by the particular print driver.
- Paper size - Allows the user to choose the paper size (letter vs. legal).
- Margins - Allows users to alter the margins for a print job.
- Retrieve Image Data At Higher Resolution (to match capabilities of the print mode) - This feature is recommended for devices that may use low-resolution displays such as televisions. If this option is not presented to the user, the device should retrieve data equal to or higher than the print resolution.

9.9.2 Warnings & Errors

The APDK has a series of warning and error codes that are used to indicate success or failure of methods. For a complete list, refer to the **APDK API** (p. 90).

Warnings (represented by a value less than zero):

A warning is code that has completely executed, but is a notification that a print concern exists. Example: A user has chosen grayscale print mode, but a color pen is installed. A warning is issued indicating that this is an inefficient use of the pen, that the printed product will not look like they expect it to, but the printer will print with it anyway unless given further direction. At this point, a dialogue box can be sent to the user via a GUI button confirming that they really want to use this setting.

Errors (represented by a value greater than zero):

An error is code that has not completely executed, and is a notification of failure and that action must be taken. Example: When a paper jam occurs during printing, a GUI should be sent to the user indicating that action(s) must be taken before the print job can continue. The APDK will return the error, but it is up to the developer to interpret that error and correct the problem or notify the user that action must be taken.

9.9.3 DisplayPrinterStatus

The method DisplayPrinterStatus is used by the core code to communicate the printer state to the host system and ultimately the end user. The Driver uses DisplayPrinterStatus to inform the host of any error conditions, which the host environment then handles appropriately.

For example, if the printer has a mechanical error, the printer flags that an error has occurred. The core code stops sending data to the printer and calls DisplayPrinterStatus. The host then flashes a GUI conveying the error to the user. The DisplayPrinterStatus method is implemented in the derived host **SystemServices** (p. 33) class.

DisplayPrinterStatus values can either be recoverable or non-recoverable, and/or display the current status of the print job. The developer should keep this in mind when creating GUIs, and be sure to include a Cancel or OK button if necessary. Some display printer status values are conditions that the user can easily correct and allow the print job to continue. The following is an example of this:

A user submitting a print job incurs an error condition because there is no paper in the paper input tray. The hosting environment will be prompted with a display printer status value of DISPLAY_OUT_OF_PAPER and accordingly prompt the user with a GUI explaining the error condition, and how to resolve it. If the user puts paper into the input tray and hits

the resume button on the printer, the display printer status value will be changed and the print job can continue normally.

For recoverable error conditions such as this, the hosting environment will need to continue looping around to check the display printer status value, giving the flexibility of the error condition being resolved on the printer.

The following list of printer statuses describes the condition, whether it's recoverable or not, and where appropriate, sample text to use in the GUI.

DISPLAY_PRINTING

Condition: Currently printing - this is a flag that says the job is currently printing.
Recoverable: N/A
Text: "Currently Printing."

DISPLAY_PRINTING_COMPLETE

Condition: Printing complete.
Recoverable: N/A
Text: "Printing Complete."

DISPLAY_PRINTING_CANCELED

Condition: Printing canceled.
Recoverable: N/A
Text: "Printing Canceled."

DISPLAY_OFFLINE

Condition: Printing is currently offline; some Hewlett-Packard printers go offline to flag that an error has occurred.
Recoverable: N/A
Text: "Your printer has gone offline due to an error. Cancel the job, check the printer, and try again."

DISPLAY_BUSY

Condition: Printer is busy.
Recoverable: N/A
Text: "Printer Busy."

DISPLAY_OUT_OF_PAPER

Condition: Printer is out of paper.
Recoverable: Yes, depending on if the user cancels the job or loads more paper and presses the resume button.
Text: "The printer is out of paper. Insert paper into the paper tray, and push the resume button on the printer."

DISPLAY_TOP_COVER_OPEN

Condition: Printer top cover is open.
Recoverable: Yes, depending on if the user cancels the job or closes the top cover.
Text: "The top cover of your printer is open. Close the top cover to resume printing."

DISPLAY_ERROR_TRAP

Condition: Printer mechanism error, this is a non-recoverable error which usually requires intervention by the user.
Recoverable: No
Text: "The printer is in an error condition. Cancel the job, check your printer, and try to print again."

DISPLAY_NO_PRINTER_FOUND

Condition: No printer found.
Recoverable: Yes, depending on if the user cancels the job. Or if there is a printer attached, powers the printer on and presses the resume button.
Text: "No printer was found. Check to make sure your printer is connected properly and that the power is on. Then push the resume button on the printer to continue printing."

DISPLAY_NO_PEN_DJ400

Condition: No pen found in the Hewlett-Packard DJ400 series printer.
Recoverable: Yes, depending on if the user cancels the job, or inserts a pen and presses the resume button.
Text: "The print cartridge is either missing or installed incorrectly. Check the printer cartridge and push the resume button on the printer."

DISPLAY_NO_PEN_DJ600

Condition: No pen found in the Hewlett-Packard DJ600 series printer.
Recoverable: Yes, depending on if the user cancels the job, or inserts a pen and presses the resume button.
Text: "The print cartridge is either missing or installed incorrectly. Check the printer cartridge and push the resume button on the printer."

DISPLAY_NO_COLOR_PEN

Condition: No color pen found on Hewlett-Packard DJ 2 pen products.
Recoverable: Yes, depending on if the user cancels the job, or inserts a pen and presses the resume button.
Text: "The color print cartridge is either missing or installed incorrectly. Check the printer cartridge and push the resume button on the printer."

DISPLAY_NO_BLACK_PEN

Condition: No black pen found on Hewlett-Packard DJ 2 pen products.
Recoverable: Yes, depending on if the user cancels the job, or inserts a pen and presses the resume button.

Text: "The black print cartridge is either missing or installed incorrectly. Check the printer cartridge and push the resume button on the printer."

DISPLAY_NO_PENS

Condition: Both pens are missing from Hewlett-Packard DJ 2 pen products.

Recoverable: Yes, depending on if the user cancels the job, or inserts a pen and presses the resume button.

Text: "The print cartridges are either missing or installed incorrectly. Check the printer cartridges and push the resume button on the printer."

DISPLAY_PHOTO_PEN_WARN

Condition: Indicates photo pen in a printer that cannot make use of it.

Recoverable: No

Text: "Photo print cartridges are not supported. Cancel the print job and insert the correct print cartridges."

DISPLAY_PRINTER_NOT_SUPPORTED

Condition: Printer not supported by driver.

Recoverable: No

Text: "This printer is not supported."

DISPLAY_COMM_PROBLEM

Condition: Communication problem with printer.

Recoverable: No

Text: "Cannot communicate with the printer."

DISPLAY_CANT_ID_PRINTER

Condition: Cannot id printer.

Recoverable: No

Text: "Cannot properly identify the printer."

DISPLAY_OUT_OF_PAPER_NEED_CONTINUE

Condition: The printer is out of paper.

Recoverable: Yes

Text: "You printer is out of paper. Insert the paper into paper tray and continue."

DISPLAY_PAPER_JAMMED

Condition: Paper Jammed or Paper Stalled.

Recoverable: No

Text: "Paper is jammed in your printer. Cancel the print job and clear the paper jam."

DISPLAY_PHOTOTRAY_MISMATCH

Condition: Phototray is engaged, but requested paper size is larger than A6 or A6/Photo size is requested, but PhotoTray is not engaged.

Recoverable: Yes
Text: "Requested media size and the phototray status do not match. Engage or disengage phototray based on the requested paper."

9.9.4 Abstract Class "SystemServices" To Be Implemented By Host System Interface

The APDK requires access to certain basic services that can only be provided by the hosting environment such as memory management and input/output. The **SystemServices** (p. 33) object is used to obtain and encapsulate these services and other utilities that the printer driver uses to process print jobs, such as InitDeviceComm, DisplayPrinterStatus, ReadDeviceID, and GetStatusInfo.

SystemServices (p. 33): System Services is an abstract base class that requires implementation of various routines by the host environment in order to compile. In most cases, this implementation is straight-forward. For example, implementing the required AllocMem function should just require mapping it to "malloc" or some equivalent; however, the host system also has the opportunity to allocate all driver memory in a single block up front (in the constructor) and then use AllocMem calls to dole out pieces from this block.

Instantiation of a **SystemServices** (p. 33) object is the first step in the driver calling sequence, and must occur before either **PrintContext** (p. 22) or **Job** (p. 20) can be instantiated.

InitDeviceComm: InitDeviceComm is a method included in the base class whose function is to ensure that communication can be established with the printer, and the printer can be identified. InitDeviceComm must be called from the derivative **SystemServices** (p. 33) constructor. All other system-specific initialization should be done in the derivative **SystemServices** (p. 33) constructor as well. An example would be setting the seed for random number generation (i.e. implementation of the GetRandomNumber method).

DisplayPrinterStatus: DisplayPrinterStatus is a method implemented in the derivative class that allows communication of the driver status (errors, conditions requiring user intervention, etc.,) to user interfaces provided by the host. For more information, refer to DisplayPrinterStatus.

ReadDeviceID: ReadDeviceID is a method implemented in the derivative class for retrieving the raw DeviceID string from the printer. ReadDeviceID is dependent on the type of communications protocol used, (1284, USB, or other methods.) If access to the printer's DeviceID is not possible due to I/O issues on the host system, have this function return BAD_DEVICE_ID to allow proper operation of the printer driver's error handling system.

GetStatusInfo: GetStatusInfo is a method implemented in the derivative class for retrieving the centronics status byte from the printer. Since this "status byte" is a fundamental element of parallel I/O and is emulated in USB, this byte should be available in most systems. If this byte is not available, simply return FALSE from GetStatusInfo to ensure proper operation of the printer driver's error handling system.

9.9.5 Parallel 1284 versus USB

If USB will be the print solution I/O method, the printer driver must be informed by setting the flag IOMode.bUSB to TRUE in the constructor of the derivative **SystemServices** (p. 33) class. The drivers produced by the Hewlett-Packard Print Driver APDK require this information due to I/O and error handling differences between USB and parallel/1284 connectivity.

9.9.6 Plug & Play versus Uni-Directional

Hewlett-Packard strongly recommends using bi-directional communication (bi-di) instead of uni-directional communication (uni-di) when implementing print solutions with this APDK. Bi-directional communication provides complete communication between the user and the printer, and enables plug and play capability, which provides seamless identification and use of Hewlett-Packard DeskJet printers without any input from the user.

If a uni-directional I/O method is the only option, the driver can still be used although the user will not have the advantage of the printer-to-host system communication for troubleshooting. In this situation, the host system will need to prompt the user to report the model they will be printing to before initiating a print job. The hosting system is responsible for storing this printer model to prevent further prompting of the user again.

In normal bi-directional communication, Initdevicecomm tries to establish contact with the printer when the **SystemServices** (p. 33) class is created. If successful, the model number is saved for later use. When the **PrintContext** (p. 22) is created, the pointer for the **SystemServices** (p. 33) class will be used to check if the model number has been previously saved. If it was, bi-directional communication has been achieved.

It is also possible for **PrintContext** (p. 22) to be created when no model has been saved. The host system should always call PrinterSelected( ) to see if a printer has been selected after the **PrintContext** (p. 22) is created. A return value of TRUE will occur if bi-di communication is operational and the printer's model has been established. If it appears that no device has been selected (FALSE), then the user should be prompted to enter

the model number, and SelectDevice should be called in order to pass on the appropriate model number.

The first possibility is when no status can be read at all from the centronics port, such as in a spooling case. The developer knows that the solution will be a true uni-directional I/O and should leave out InitDeviceComm from the derived **SystemServices** (p. 33) class. The second possibility is when the hosting system would still have access to the centronics register. If the hosting environment has access to the centronics port, the register can be passed to GetStatusInfo, allowing the driver to process the status.

There are four paths for the communication model:

- No DevID, No Status
- No DevID, Yes Status
- Yes DevID, No Status
- Yes DevID, Yes Status

Where DevID indicates access to the printer's deviceID string and Status indicates access to the centronics status byte. The InitDeviceComm method looks at the return values of ReadDeviceID and GetStatusInfo, and sets the error handling functionality appropriately.

9.10 APDK Supported Printers

9.10.1 Printers Currently for Retail and Legacy Devices

- Deskjet 350C
- Deskjet 400, 400L
- Deskjet 450, 460 Series
- Deskjet 500 Series, 520, 540
- Deskjet 6XX series
- Deskjet 810C, 812C, 815C, 816C, 830, 832, 840, 841, 842, 843
- Deskjet 825, 845
- Deskjet 850, 855, 870, 880, 882, 890, 895
- Deskjet 9XX Series
- Deskjet 11XX Series, 12XX Series
- Deskjet 3810, 3816, 3818, 3819, 3820 Series, 3822, 3870
- Deskjet 51XX, 54XX, 55XX, 56XX, 57XX, 58XX, 59XX
- Deskjet 61XX, 65XX, 66XX, 68XX, 69XX
- Deskjet 5600 & 5100 & 5800
- Deskjet 6540 & 6520
- Deskjet 5740
- Deskjet 6840
- Deskjet 3320 & 3420 & 3325
- Deskjet 3600 & 3500
- Deskjet 3740 & 3840
- Deskjet 9300
- Deskjet 96XX
- Deskjet 98XX

- Deskjet D25XX
- Deskjet F42XX
- Officejet 3XX, 5XX, 6XX, 7XX
- Officejet 11XX
- Officejet 51XX, 61XX, 62XX, 63XX, 4100, 4105, 4200, 5500, 9100
- Officejet 71XX, 72XX, 73XX, 74XX
- Officejet 9x11
- Officejet d Series
- Officejet G Series
- Officejet J Series
- Officejet K Series
- Officejet Lx
- Officejet R Series
- Officejet t Series
- Officejet v Series
- Officejet Pro K Series
- Officejet Pro L Series
- Officejet H470
- Officejet J35XX, J36XX, J55XX, J57XX, J64XX,
- Officejet K71XX
- PSC 500, 7XX, 9XX
- PSC 15XX, 16XX
- PSC 21XX, 22XX, 23XX, 24XX, 25XX
- PhotoSmart 1XX, 2XX, 3XX, 4XX
- PhotoSmart 11XX, 12XX, 13XX
- PhotoSmart 25XX, 26XX, 27XX
- PhotoSmart 31XX, 32XX, 33XX
- PhotoSmart 71XX, 72XX, 73XX, 74XX, 75XX, 76XX, 77XX, 78XX, 79XX
- PhotoSmart 80XX, 81XX, 82XX, 84XX, 87XX
- PhotoSmart A3XX, A4XX, A5XX, A6XX, A7XX, A8XX
- PhotoSmart C31XX, C41XX, C42XX, C43XX, C44XX, C51XX, C52XX, C61XX, C62XX, C71XX, C72XX, C81XX
- PhotoSmart D50XX, D51XX, D53XX, D54XX, D61XX, D71XX, D72XX, D73XX, D74XX
- PhotoSmart Pro B83XX
- PhotoSmart P1000, P1100
- All Monochrome Laserjet Printers and MFPs (except LJ 1000, 1005, 1018, 1020, P10XX, M1005)
- All Color LaserJet Printers and MFPs (except LJ 1600, 2600)
- Apollo P2100 & P2150
- Apollo P2200 & P2250
- E-Printer e20
- Business InkJet 1000, 1100 Series, 1200 Series, 22XX Series, 2300 Series, 2600 Series, 2800 Series, 3000 Series
- Color Inkjet cp1700 Series
- HP 2000 Series, 2500 Series
- Apollo 2200, 2500, 26XX, P2XXX

For current pricing, please visit `http://www.hp.com`

For more information on printers, features, and APDK support, refer to **Printers and Features Supported** (p. 86).

9.11 Printing

The Hewlett-Packard Appliance Printing Software Development Kit (APDK) is designed to offer basic printing support for "thin client" environments. As such, the breadth of available features is secondary to supplying optimal speed and print qualities over several printer families, while maintaining a small code-space/ run-time footprint.

The below topics discuss printing and the APDK in greater detail.

9.11.1 Printers and Features Supported

The following features are supported by the APDK and apply to all printer families and family members:

- 24-bit RGB input or 24-bit RGB and 1 Bit Black input.
- Multiple Paper Sizes (Letter, A4, Legal)
- Grayscale (Black-Only) Mode
- Comprehensive Error Handling & Notification

Currently supported printer families, their family members, and particular features available are listed in the following table. Printer family derivatives are available on the market and should be considered to have similar feature functionality as other members of their family type.

Note:

8xx Series Only - The scope of this Series does NOT include the older 850C, 855C, 870C or 890C printers.

Printer Family / Class / Model Functionality Table

Table still needs to go here

N/A - This feature does not come with this printer.

NS - No Support. This feature will be ignored or actively return an error.

The following Hewlett-Packard DeskJet printers are not supported by the APDK:

- 400 series
- 700 series
- 850C, 855C, 870C or 890C

Note:

Unless explicitly stated, all non-Hewlett-Packard Printers and all Hewlett-Packard non-DeskJet Printers are not supported by this APDK.

9.11.2 Minimum Functionality

Print Quality

The quality of printed output can vary widely in the thin-client environment, due to host limitations. Some factors that can affect quality are:

- The presence of a scalable font engine in the thin client host.
- The use of internal printer fonts where possible.
- Composite black printing on single pen printers (DJ400, DJ600).
- The quality of document to be printed.
- The quality, scaling, and compression of document images.
- Anti-aliasing and other algorithms designed to make text look good on a TV screen.

During testing, it is important to keep these limitations in mind.

Print Modes

The firmware in most DeskJet printers is written to accept various kinds of tradeoffs between speed and print quality, and to optimize output for different types of paper and ink cartridges (pens). The differences involve a complicated variety of parameters including resolution, number of inks, and special printer commands; for each printer model, only a small handful of such combinations are allowable. The APDK packages these combinations inside a structure called a Print Mode. The Print Mode hides technical details and presents the packaged combinations in such a way that they can be concisely related to the conditions for which they were designed.

The number of print modes for each printer type is given by the GetModeCount function in **PrintContext** (p. 22). Typically, there is a default color mode and a grayscale mode; other modes with special properties such as high-resolution photo quality are available for certain printer models. In some cases, the developer must choose between features such as device- based text and photo quality. This is accomplished by selecting the mode (Print Context member SelectPrintMode,) and then querying with PrinterFontsAvailable.

Deprecated:

Every mode has a string identifier (for potential use in a user interface) given by GetModeName. **For version 3.0 and above this statement is no longer true.** GetModeName is deprecated and always returns an empty string. It remains in the API for backward compatibility - new code should not use it and old code should be converted.

When bidirectional communication with the printer is available, the **PrintContext** (p. 22) constructor selects a default mode based on the

installed pen. (The default mode is the normal quality based on plain paper that is compatible with the pen.) In order to override this setting, (possibly due to selected user preferences or unavailable bidirectional communication,) the SelectPrintMode method is provided in **PrintContext** (p. 22).

Deprecated:

SelectPrintMode takes a zero-based index. By convention, index 0 always refers to the Grayscale mode GRAYMODE_INDEX=0). Index 1 DEFAULTMODE_INDEX refers to the "normal" color mode; other values up to <GetModeCount-1> refer to specialized modes. If the selected Print Mode parameter requested is incompatible with the installed pen, SelectPrintMode returns a WARNING. Because it is a warning and not an error, it is possible to proceed using the requested mode even under the incompatible conditions. Examples of incompatibilities include using GRAYMODE when the pen has colored ink. The results are a gray-looking output that was wastefully produced by combining colored inks. Another incompatibility, using DEFAULT MODE when a "photo-pen" is installed, results in "normal" pen output quality instead of output the more expensive photo-pen was intended to deliver. Some incompatibilities may even result in no useful output whatsoever.**As of version 3.0 this API is retained (temporarily) for backward compatibility. New code should not use this API, and existing code should migrate to the new SelectPrintMode.**

SelectPrintMode (p. 31) takes 4 parameters: they are quality, media, color, and fonts. This allows for more precise control over the print mode selected and provides for more print modes for specific needs. There are many more combinations then there are print modes, but SelectPrintMode will choose the mode that most closely matches the request. SelectPrintMode will return a mismatch warning if the requested parameters cannot be accommodated exactly with the printer, however SelectPrintMode will have selected the most appropriate mode to match the request. If a mode mismatch occurs then GetPrintModeSettings() can be called to determine the values of quality, media, color, and fonts that were set. If they are acceptable then progress should continue, if not then SelectPrintMode can be called again with a different set of parameters.

Note:

GetPrintModeSettings can be called to determine the mode values and the glue code can use these values to determine print mode names or strings that can be used in a user interface. This method should be used instead of GetModeName.

Changing the Print Mode may also change the Effective Resolution; therefore, resolution should be queried again after changing the Print Mode. (In

general the default/normal modes will all use an Effective Resolution of 300 DPI, while certain best or special modes may use 600 DPI or higher) For older printer models, the Print Mode may also determine whether printer fonts are allowed or not.

Resolution

Effective graphical output resolution depends on the PrintMode selected for the attached Printer model.

To determine the acceptable resolution for the currently selected printer and mode, call the **PrintContext** (p. 22) function, EffectiveResolution. Remember that the calls must be in the appropriate order, or the results will not be valid. For example, if SelectDevice and/or SelectPrintMode are called after EffectiveResolution, the result of the EffectiveResolution call will no longer be valid.

Scaling

When the graphical source data exists at resolutions lower than Effective-Resolution, the data must be scaled up to reach the desired resolution. The APDK code does this automatically, based on the relation between the two values InputPixelsPerRow and OutputPixelsPerRow - the two parameters to the **PrintContext** (p. 22)'s constructor or SetPixelsPerRow function.

When the graphical source data exists at resolutions higher than Effective-Resolution, the host must sub-sample in order to reach the desired input resolution. The APDK code provides no assistance in this regard - in other words it is illegal to set InputPixelsPerRow greater than Output-PixelsPerRow.

Note:

It is always preferable to obtain original data at a higher resolution and sub-sample the data, rather than to take previously sub-sampled data and scale it up.

For more information, refer to the Best Practices for Optimal Printing, **Image/Text Resolution** (p. 70).

9.12 APDK API

The Hewlett-Packard Appliance Printing Software Development Kit (APDK) API reflects the "lean" footprint and memory usage required for an applicance driver. It is not immense, and should not take long to become familiar with.

The API is broken into three sections. First, the API gives a brief description of the four public classes used to run the driver: **SystemServices**

(p. 33), **PrintContext** (p. 22), **Job** (p. 20), and **Font** (p. 18). (Not each of these classes will be used for every implimentation.) After the descriptions are tables with class members, a one or two-sentence description, and any return type(s) or parameter(s), if applicable. Following the tables are detailed paragraphs for each member, with parameters, return values, and error codes.

9.12.1 API Overview

Global Functions

There is one Global Function, Version used to determine properties of the current version. All other API calls are made through one of four C++ objects.

Object Classes

There are four publicly documented object classes. Not all of these classes will be used in every host system. They are:

- **SystemServices** (p. 33) - This object encapsulates functionality that must be supplied by the host system, including memory management and I/O, clock access, user notification, etc. The abstract base class must be instantiated in a host-specific derived class, where the system code is hooked up to the required driver entry points. The only reference to this object in API-calling code should be to its constructor and destructor, which must be invoked prior to and after all other APDK calls.
- **PrintContext** (p. 22) - This object serves two purposes. First, it encapsulates knowledge about all devices supported in available components. It can be queried at runtime to determine capabilities of the attached device. Second, it allows for setting optional parameters of a print job, such as media size and margins, use of color and data-resolution, etc.
- **Job** (p. 20) - This object receives processes and transmits data to be printed by one device on successive pages. The **PrintContext** (p. 22) and **Job** (p. 20) together form the core of the APDK code.
- **Font** (p. 18) - This object is used to control printer fonts when sending ASCII data. (For systems that lack a reasonable font engine or for some reason can benefit from using printer fonts.)
- Subclass **ReferenceFont** (p. 33) (EnumFont) is used to query available font properties prior to instantiating the font. Whereas **Font** (p. 18) objects created upon request are to be deleted by caller, ReferenceFonts live with the core structures and cannot be deleted.

Auxiliary Objects

– Scripter - For debugging purposes only. This object provides script recording and playback functionality. (See the **SystemServices** (p. 33) functions InitScript and EndScript.) It comes in two flavors, ASCII and binary. A platform that lacks a file system can add an additional subclass using terminal-emulation captures, or whatever means is available.

9.13 Sample Code

The Hewlett-Packard Appliance Printing Software Development Kit (APDK) presupposes a bidirectional connection with the printer and the use of defaults for all optional settings. It also presupposes that the graphical data is available at 300 dpi, and the host does that rendering of text. With these assumptions in place, the below are examples of API calls in sample code.

9.13.1 Minimal Sequence

This minimal sequence is presented in the form of a routine that handles one print job, and returns an error-code to the caller. The number of pages and page height are given, as well as the graphical data. (For demonstration purposes, this is imagined as a giant array.) The commented numbers on the right correspond to notes about the code at the end of the code sequence.

```
#include "hpprintapi.h"                                    //1

using namespace APDK_NAMESPACE;                            //2

DRIVER_ERROR PrintJob
(
    unsigned int NumPages,
    unsigned int PageHeight,                               //3
    unsigned int PageWidth,                                //4
    BYTE JobArray[NumPages][PageHeight][PageWidth*3]       //5
)
{
    PlatformServices* pSys = new PlatformServices();       //6
    if (pSys==NULL)                                        //7
    {
        return SYSTEM_ERROR;
    }

    DRIVER_ERROR err = pSys->constructor_error;            //8
    if (err > NO_ERROR)                                    //9
    {
        return err;
    }
    else if (err < NO_ERROR)
```

```
{
    // warning - dialog with user
}

PrintContext *pPC = new PrintContext(pSys, PageWidth);  //10
if (pPC == NULL)
{                                                       //11
    delete pSys;
    return SYSTEM_ERROR;
}

err = pPC->constructor_error;
if (err > NO_ERROR)
{
    delete pSys;
    return err;
}
if (err < NO_ERROR)
{
    // warning - dialog with user
}
Job* pJob = new Job(pPC);                               //12
if (pJob == NULL)
{
    delete pPC;
    delete pSys;
    return SYSTEM_ERROR;
}

err = pJob->constructor_error;
if (err > NO_ERROR)
{
    delete pPC;
    delete pSys;
    return SYSTEM_ERROR;
}
else if (err < NO_ERROR)
{
    // warning - dialog with user
}

// JOB LOOP
for (int i=0; i < NumPages; i++)
{
    // PAGE LOOP
    for (int j=0; j < PageHeight; j++)
    {
        err = pJob->SendRasters(JobArray[i][j]);        //13
        if (err > NO_ERROR)
        {
            delete pJob;
            delete pPC;
            delete pSys;
            return err;
        }
    } //PAGE LOOP
```

```
        err=pJob->NewPage();                                    //14
        if (err > NO_ERROR)
        {
            delete pJob;
            delete pPC;
            delete pSys;
            return err;
        }
    } //JOB LOOP

    delete pJob;                                                //15
    delete pPC;
    delete pSys;
    return NO_ERROR;
} //PrintJob
```

Notes on Minimal Sequence

1. "hpprintapi.h" is the only file that needs to be included to interface with the APDK core code.

   ```
   #include "hpprintapi.h"                                      //1
   ```

2. All names in the APDK are in the **apdk** (p. 18) namespace. Immediatly after the inclusion of "hpprintapi.h" use a using namespace APDK_NAMESPACE; to include the **apdk** (p. 18) namespace in the compilation.

   ```
   using namespace APDK_NAMESPACE;                              //2
   ```

3. PageHeight is given in pixel-rows. For a default page with a vertical printable area of 10 inches and effective resolution of 300 dpi, this value would be 3000.

   ```
   unsigned int PageHeight,                                     //3
   ```

4. PageWidth is given in pixels per row. Note that this value is constant across the **Job** (p. 20); it must not exceed the maximum width for the target paper- size. (In this example using defaults, this would be assumed to be 2400 for LETTER size paper with 8 printable inches at 300 dpi) All rows of data must be padded with white space (RGB value 0xFFFFFF).

   ```
   unsigned int PageWidth,                                      //4
   ```

5. The size of a row in the byte-array is given as PageWidth∗3, because each pixel is represented as 3 bytes in the required RGB24 format.

   ```
   BYTE JobArray[NumPages][PageHeight][PageWidth*3]             //5
   ```

6. Creation of a **SystemServices** (p. 33) object is the first time communication with the printer is attempted, and entry points required by the core code are mapped. The class **SystemServices** (p. 33) is abstract, so reference is made here to the derived class containing the implementation for this platform.

   ```
   PlatformServices* pSys = new PlatformServices();             //6
   ```

7. Check that operator new succeeded.

```
if (pSys==NULL)                                          //7
{
    return SYSTEM_ERROR;
}
```

8. The code set presumes no throw/catch capability in the local compiler, so all errors in constructors are signaled by the value of constructor_error, which the caller must check.

```
DRIVER_ERROR err = pSys->constructor_error;              //8
```

9. "Hard" errors have values greater than zero (NO_ERROR). Values less than zero are warnings, which permit execution to continue; however, the user should be alerted to them.

```
if (err > NO_ERROR)                                      //9
```

10. The second step involves setting up the **PrintContext** (p. 22) for the job, which contains all the information about the print modes, paper type and size, etc. The constructor sets defaults for all necessary options, and defines the printer model being used based on information gathered by **SystemServices** (p. 33) constructor.

```
PrintContext *pPC = new PrintContext(pSys, PageWidth);  //10
```

11. When exiting due to an error, this step cleans up all previously allocated objects.

```
{                                                        //11
    delete pSys;
    return SYSTEM_ERROR;
}
```

12. The **Job** (p. 20) object handles one continuous sequence of rasters on successive pages. Data will be sent through this object.

```
Job* pJob = new Job(pPC);                                //12
```

13. All the data is sent here. Errors may include conditions such as printer out of paper, offline, etc.

```
        err = pJob->SendRasters(JobArray[i][j]);         //13
```

14. This function causes the page to be ejected, and resets various internal counters.

```
        err=pJob->NewPage();                             //14
```

15. This step cleans up all allocated objects.

```
    delete pJob;                                         //15
    delete pPC;
    delete pSys;
    return NO_ERROR;
} //PrintJob
```

9.13.2 Using ASCII Text

The availability of font types varies by printer model and print mode. Up to four font types are contained in the APDK code set: Courier, LetterGothic, CGTimes, and Univers. Once the printer model has been selected (implicitly or explicitly), and a print mode decided upon, use the **PrintContext** (p. 22) method PrinterFontsAvailable to determine whether ASCII support is available at all. If it is, use the method EnumFonts to iterate through the set of ReferenceFonts, querying each for desired properties such as fixed-pitch vs. proportional. Then obtain a **Font** (p. 18) object for use in TextOut calls, by means of the method RealizeFont.

The following example shows a minimal sequence in which default settings are used for all the properties including color, bold, underline, and italics. The first step is to determine whether printer fonts are available on the current printer, using the currently selected mode. (This example assumes nothing has been done about setting PrintModes yet.)

1. Fetch the mode that has been automatically set:

   ```
   int ModeIndex = pPC->CurrentPrintMode();
   ```

2. Then query for font support:

   ```
   BOOL fontsOK = pPC->PrinterFontsAvailable (ModelIndex);
   ```

3. If fontsOK is FALSE, cause the system to render the fonts into the graphics buffer. Otherwise, create a **Font** (p. 18) object to use for sending ASCII text.

   ```
   // create Courier font at 12-point size
   pFont = pPC->RealizeFont( COURIER_INDEX, 12);
   err = pFont->constructor_error;
   if (err > NO_ERROR)
       // try again with different parameters, or bail out
   ```

Multiple fonts may of course be created for use on different portions of the text; it is up to the caller's layout code, however, to determine sizes and locations on the page. Other **Font** (p. 18) methods such as GetTextExtent may also be used in the layout code. The text may be sent at any time while processing a page. Each string of characters using the same font and running continuously across a row should be sent in one call to **Job** (p. 20):

```
err = pJob->TextOut("Hello World", strlen("Hello World"), *pFont,
x,y);
```

Warning:

Errors may result from sending text that would run off the page, in addition to error conditions in the printer.

9.13.3 User-Interface Calling Code Samples

The following functions demonstrate how to relay information to the user about actual and potential printer models connected to the system, and about available print modes.

The three examples are presented as independent functions, and therefore each has the overhead of creating and deleting the **SystemServices** (p. 33) and **PrintContext** (p. 22) objects. In your code, these objects may already exist at the time this information is needed.

Note:

Strings: although basically in English, the strings used by these functions have been formulated so as to present minimal translation difficulties. Printer model names have the word "DeskJet" followed by numbers. Print mode names are no longer supported.

Example 1: Show User What Printer Models Are Supported In The Current Build

This example is a function to return the number of supported model families and an array of their names. This functionality is only required when the connected printer cannot be identified directly through hardware, and thus the user must tell the system which model to target. This could occur because of a faulty printer cable, or in systems that direct output across a network or spooling system.

```
DRIVER_ERROR AvailableModels
(
    char* modelarray[],                                          //1
    unsigned int& count
)
{
    PlatformServices* pSys = new PlatformServices();
    if (pSys==NULL)
    {
        return SYSTEM_ERROR;
    }

    DRIVER_ERROR err=pSys->constructor_error;
    if (err > NO_ERROR)
    {
        return err;
    }

    PrintContext *pPC = new PrintContext(pSys);
    if (pPC==NULL)
    {
        delete pSys;
        return SYSTEM_ERROR;
    }

    err=pPC->constructor_error;
    if (err > NO_ERROR)
    {
```

```
        delete pSys;
        return err;
    }

    count=0;
    FAMILY_HANDLE  familyHandle = NULL;
    PRINTER_TYPE p = pPC->EnumDevices(familyHandle);
    while (p != UNSUPPORTED)
    {
        modelarray[count++] = (char*)pPC->PrintertypeToString(p);
        p = pPC->EnumDevices(familyHandle);
    }

    delete pPC;
    delete pSys;
    return NO_ERROR;
} //AvailableModels
```

Notes on Available Models

1. The caller of this routine will have allocated an array of size MAX_-PRINTER_TYPE+1. MAX_PRINTER_TYPE is the index of the highest PRINTER_TYPE enum in a zero-based count.

```
        char* modelarray[],                                    //1
```

2. EnumDevices is an iterative mechanism for listing each supported model in the build. It returns an enum representing the printer model family.

3. PrintertypeToString converts the enum to a string suitable for display to user.

Example 2: Show user the specific model name of the connected printer

The model names and enums used in Example (1) above refer to printer-model families. These families share all important firmware and pen types and are treated as a single type from the perspective of the APDK code. But further differences are found within these families. For example the family that includes the 690 models has members with particular designations such as 692. This detailed information, matching the designation on the customer's device and packaging, exists in the device memory, and can be obtained through the following method.

```
DRIVER_ERROR GetConnectedModel
(
    PRINTER_TYPE& pt,
    const char* ModelIDStr
)
{
    PlatformServices* pSys = new PlatformServices();
    if (pSys==NULL)
```

```
    {
        return SYSTEM_ERROR;
    }

    DRIVER_ERROR err=pSys->constructor_error;
    if (err > NO_ERROR)
    {
        return err;
    }

    PrintContext *pPC = new PrintContext(pSys);
    if (pPC==NULL)
    {
        delete pSys;
        return SYSTEM_ERROR;
    }
    err=pPC->constructor_error;
    if (err > NO_ERROR)
    {
        return err;
    }
    pt = pPC->SelectedDevice();                          //1
    ModelIDStr = pPC->PrinterModel();                    //2
    delete pPC;
    delete pSys;
    return NO_ERROR;
} //GetConnectedModel
```

Notes on GetConnectedMode example

1. The device is selected explicitly here; as with the rest of the "boilerplate", this work may have already been done by the time this functionality is needed.

   ```
   pt = pPC->SelectedDevice();                          //1
   ```

2. Fetches the model number string from printer memory.

   ```
   ModelIDStr = pPC->PrinterModel();                    //2
   ```

9.14 Support & Debugging

9.14.1 Developer

General Support

All types of support by HP for APDK software are discontinued.

9.14.2 End-User

General Support

All types of support by HP for APDK software are discontinued. Support responsibility for the APDK as part of any product lies with the Developer of the product.

HP printer specific help can be accessed by use of a standard PC based browser at `http://www.hp.com`.

iTV Support

Customers using iTV (interactive TV) and the APDK to print, and encountering a problem will see an on-screen error message to help them resolve the issue. If this level of self-help is not sufficient the customer has the option of clicking the on-screen *more* button that provides access to a HP hosted server, which serves up correctly formatted iTV printer specific On-Screen-Help.

9.14.3 Support Responsibility

Hewlett-Packard charges no fees for the open source APDK core code and its associated documentation. However, support responsibility for the APDK lies with the Developer using it. All types of support for APDK are discontinued by HP.

For purposes of debugging, device manufactures using the APDK have the ability to have their device return the Version Number string from the core APDK code as well as to capture APDK code-base related information to a log file. HP encourages APDK developers to implement such functionality and to inform their support staff and customers on how to use it.

9.15 HP-APDK Driver Debug Procedure (High Level)

The following procedure describes how the HP APDK can be debugged. The idea is to simply isolate the problem by a process of elimination between Hardware and Software down to the root cause level.

9.15.1 Hardware Debugging

Differentiate between the *Sending Device* and *Receiving Device* being the cause of the problem by doing a *Sender Self Test* and a *Receiver Self Test*. Ensure the receiving device is an APDK supported Printer.

If the *Sending Device* hardware is identified as defective, re-direct the customer to that device manufacturers' help desk. If the *Sending Device* is not defective, then ensure that the *Receiving Printer* hardware is fully functional.

Next eliminate the *Transport* (USB, parallel are examples of transports) as the source of the problem by ensuring that the cable or connection is solid, and the associated software for the corresponding transport has been

correctly installed and is active. If the physical transport and its associated software is problematic then direct the customer to the appropriate manufacturers support desk for that transport/device.

9.15.2 Software Debugging

Once you have determined that the devices' hardware functions normally then isolate the problem to the sending devices' driver by doing a *Printer Driver Test*. For example in a Windows environment, perform the following - Start >Settings > Printers > Choose the Printer > Properties > Test Page Button, or in the case of a non-Windows device follow the equivalent manufacturer suggested test method for the device or application.

Next, identify the device *Driver Manufacturer* as HP or Non-HP and determine the type of driver being used i.e. Is it a regular *product specific HP released driver* that shipped with the printer or a APDK *developers driver*? If the driver being used is a HP product specific released (not open source) driver and is defective then direct the customer to that HP devices' driver support team. See Note below, if the steps in this paragraph and the next one cannot be accomplished.

On the other hand, if the driver is a HP APDK open source driver, as evident from the *Driver ID*, then an *APDK Core Code Test* should be performed on the device or application. Manufacturers may or may not document these two steps in an easy-to-use, end-user executable manner. If this were to be the case, the customer should contact the device manufacturer and ask for these steps to be performed on an identical unit. If the APDK core code is problematic the developer of the application could be contacted. HP no longer provides any type of support for APDK.

However, if the APDK Core Code Test passes, it indicates that the HP owned portion of the APDK code-base is probably not faulty, but rather the glue code surrounding it is (as developed by the device manufacturer) in which case the customer should be encouraged to work with the support organization of the device manufacturer or developer who wrote the glue code for the sending device, in order to resolve the matter.

Note:

For purposes of debugging, device manufactures using the APDK have the ability to have their device return the Version Number string from the core APDK code as well as to capture APDK code-base related information to a log file. HP encourages APDK developers to implement such functionality and to inform their support staff and customers on how to use it.

9.15.3 Design for Support Recommendations

Develop a method to clearly identify the printer driver as being APDK based. *Use the* **Version***()* (p. 10) *function to help accomplish this.*

Develop a Core Code Self Test. *This functionality has been made available but requires the developer to run the application with a log-file-enabled driver to capture the output, most likely at the developer site or possibly in a manner that can be accomplished by the end user of the device or application using the APDK.*

Publish the list of APDK Tested and Supported Devices and associated support contacts/numbers.

9.15.4 HP-APDK Core Code Specific Debug Procedure

Hewlett-Packard has learned through experience that it can be very difficult to disentangle pieces of a printing system from other applications and operating system items such as user-interfaces, memory management, network access, rendering of images and document formatting. As a result, finding the cause of undesirable behavior or output often leads a developer through all these areas.

In contrast to this complexity, the core code delivered in the Hewlett-Packard Appliance Printing Software Development Kit (APDK) is functionally very simple: it can only take graphical, and in some cases, text data and relay it to a printer in the native format of that printer, reporting the status of this communication back to the calling code. This functional limitation does have an advantage in that the interface to the core code can be very well defined and the internal behavior easily understood.

Given a specific set of inputs to the API, Hewlett-Packard (HP) has created a script that can easily determine whether the behavior of the core code is as expected or not, and consequently whether any problems should be sought elsewhere. In addition, it is often possible to determine the mistake in the upstream code just by looking at the calling sequence or data.

Debugging a APDK based driver with Scripts described in the APDK documentation illustrates how to use this script to debug.

The APDK has been instrumented so that when a debug flag (build define "APDK_CAPTURE" is set, all API calls along with their data are recorded in what Hewlett-Packard calls a script. From this script, a large class of problems can be immediately found that would otherwise potentially take a very long time to debug.

9.15.5 Recording a Script

Developers can utilize the script recording device when debugging, problem solving, or quality testing.

The first step in producing a script is to decide on the mechanism for storing it, and the format to be used. The easiest way to produce a script is to write the data to a file on local storage using standard "fopen/fwrite" commands. If such storage is available, the code included in the APDK will be sufficient. The data can be written in either binary or ASCII form; the binary form will produce smaller files and run faster.

Hewlett-Packard has encountered some systems where a storage disk is not available and the only means of capturing data is to use printf commands that are sent to a terminal window. In this case the base-class scripting code has to be overridden (in the derivative **SystemServices** (p. 33) class). The code can be copied as is from the base-class routines, replacing every file-writing call (fwrite, fputc, etc.) with a display call (printf), and avoiding the fopen/fclose calls in the methods OpenDebugStreamR and CloseDebugStreamR. In order to capture the data from the terminal window, the ASCII format must be used. (This is why the ASCII option has been provided.) Thus the code that needs to be overridden is in the AsciiScripter subclass of Scripter, found in the file script.cpp.

The second step is for the calling code to invoke the **SystemServices** (p. 33) methods InitScript and EndScript. InitScript, should be called after the **SystemServices** (p. 33) object is constructed and before any other API calls are made. InitScript takes two parameters: the name of the script file to be created on disk (ignored in the diskless implementation), and a flag that is TRUE if ASCII format is to be used. EndScript should be called after everything else has finished and been cleaned up, except of course for destruction of the **SystemServices** (p. 33) object. Both these calls should be conditionally compiled into the calling code, so that they only occur in debug builds.

The final step is to compile all the code with the symbol APDK_CAPTURE defined, and with the files in the "debug" subdirectory included in the build. Note that APDK_CAPTURE should not be defined in any production builds, as performance will be severely degraded.

Run the test print-job and resulting file (one per job - managing the files as you see fit through passing in different filenames, or copying the result to a different filename after each run) can be used for troubleshooting. HP no longer provides any type of support for APDK.

9.16 Low Level I/O

This section provides information to begin understanding low-level I/O support and the Hewlett-Packard Appliance Printing Software Development Kit (APDK).

9.16.1 Low-Level I/O Support

Creating a customized printer driver for a specific device or platform does not necessarily require any detailed knowledge of the underlying I/O (IEEE 1284 or USB). The printer driver depends on the underlying Operating System (OS) to provide basic access to whatever I/O port is connected to the printer.

One of the first steps in porting the printer driver code is to create the derived **SystemServices** (p. 33) object for a specific device or environment. Once in place, the core code simply calls the **SystemServices** (p. 33) object for access to the I/O. If other printers are already supported by the printer driver, the necessary services that the new OS requires should be readily available.

If no printing port currently exists for the host system, additional work will be required by the developer. The details required to add the necessary I/O hardware and I/O driver support are beyond the scope of this APDK, and are not included; some basic information has been included that should help the developer begin the process.

9.16.2 Adding Parallel or USB Hardware

If third party chipsets are being considered for adding I/O capability, remember that not all chipsets are of equal quality. Be sure to choose a reputable vendor and always make certain that adequate testing occurs early between the host system and as many different peripherals as possible.

If the host system already has USB support but it has not been used for printing, make sure to test the bulk transfer' capability, which is required for printing. (Many USB peripherals (such as mice, keyboards, joysticks, etc.) do not use the bulk transfer method and therefore prior testing might not have been adequate.) In addition, some older USB controller chipsets produce an error when a high-speed device such as a printer or a scanner, and a low speed device such as a keyboard or a mouse are used at the same time. Hopefully, the low-level I/O drivers will be available from either the chip vendor or from your OS vendor. If this is not the case, then obtain the necessary industry standards documents and implement the I/O driver(s).

9.16.3 USB Documentation Sources

Additional documentation on USB may be accessed at `http://www.usb.org/developers/docs.html` , in which the official "USB Specification Document" can be found. The official "Printer Device Class Document" is located at

http://www.usb.org/developers/devclass_docs.html#approved.
As of this writing, the current "USB Specification" is version 2.0 and the "Printer Device Class Document" is 1.1. Implementing USB will rely on both of these documents, with cover-to-cover knowledge of the Printer Device Class Document, and with reference to the much larger and more generic "USB Specification Document" as necessary to implement the USB protocol stack.

9.16.4 Parallel / IEEE 1284-1994 Documentation

HP printers work off the 1284 IEEE Specifications. No additional specifications beyond these have been added. As such, there is no document describing what features of the 1284 spec are implemented for each Hewlett-Packard printer model. Regardless of price and performance, all Hewlett-Packard DeskJet printers use the same 1284 capabilities, and do not utilize RLE. (See Glossary for further explanations of these two terms.) Therefore, ECP/forward mode requirements should be taken into account when adding 1284 hardware.

Centronics compatibility mode is required for output to the printer. "Nibble" mode is required to retrieve the printer's DeviceID. See http://www.ieee.org for official documentation.

9.17 Testing

Hewlett-Packard has tested the Hewlett-Packard Appliance Printing Software Development Kit (APDK) code prior to its release. As such, the developer does not need to test the APDK code, but rather their implementation of the printer driver using the APDK code set. Hewlett-Packard suggests using 3 printer models to thoroughly test the printer driver: the 640, the 840, and one printer from the 900 family series. The suggested testing can be broken into the below testing categories.

9.17.1 Compatibility Testing

During the development cycle, developers are responsible for testing their implementation of the printer driver for compatibility with their host systems. In situations where it makes sense, the testing should include:

- **Setup** - Connecting the printer to the host system, making sure it is recognized, and can be printed to.
- **Output correctness** - Formatting, page-break handling, text/graphics overlay.
- **Printscreen** - This may be optional. In addition to the normal prints, it is important to do a printscreen if the device supports it.
- **Supported paper sizes** - Letter, A4, Legal, Photo.

- **Printing configurations** - Exercising available UI controls for printing.
- **UI Controls** - Verify all UI controls are working properly; i.e. Print Mode sets the proper output mode, warnings and errors are communicated properly, etc..
- **Memory resources** Confirm that memory is being cleaned up after print jobs complete, print jobs are canceled, etc..

9.17.2 Error Handling Testing

Error Handling testing checks the driver's response to unexpected external events. Testing should include the following user scenarios, of which there are two possible user responses: the user cancels the job when an error dialog appears, or the user rectifies the error condition and presses the printer power button to resume printing:

- Printer not connected at print time.
- Printer not powered on at print time.
- Printer powered off during a print operation via printer power button.
- Printer powered off during a print operation via power source.
- Printer cable disconnected during a print operation.
- Cover open at start of print operation.
- User lifts cover during a print operation.
- Missing ink cartridge. Test both cartridges for two-pen products, and make sure that the correct pen is reported as missing.
- Canceling print jobs. Make sure resources are cleaned up and the user can start a new print job without problems, etc. (Testing should include canceling a job both before and after printing begins. For printers that have an on-printer cancel button, this should also be tested both before and after printing begins.)
- Out of paper. Test condition where printer is out of paper before print job starts and where printer runs out of paper during multi-page print job.
- Paper jam in mechanism. Simulate by using banner paper, (taping four sheets of 8.5" X 11" paper together works as well) and printing a multi-page document.
- Photo tray detected for printers that support it.

9.17.3 Print Quality Testing

Print quality testing ensures that the driver is producing appropriate printed output. Testing should include at least the following types of documents:

- Satisfactory output printing of text documents, mixed text & graphics documents, and graphics documents.
- Both single and multi-page documents.

- Satisfactory output printing in the different modes - grayscale, normal and photo.
- No artifacts should be present in the printed output. Artifacts may include:
  * Clipping
  * Missing lines, graphics
  * Unexpected objects
  * Significant color shifts from original document source
  * Banding

9.17.4 Performance Testing

Performance testing ensures that the driver is delivering the printed output in an acceptable time. Hewlett-Packard has provided documents for the Performance tests, which can also be used for Final Validation testing. Performance tests should be run with a non-debug driver.

The following table gives the name of each file and the average time it should take to print on each printer series. Times are measured from the moment the print process is initiated until the paper drops into the output tray.

9.17.5 Final Validation

The Final Validation testing step verifies the final printer driver product by using a document set that exercises different parts of the CALLING code (browser, formatting, etc.). For final validation testing, developers should utilize the same test documents used in Performance Testing, and should run the tests with a debug driver.

The APDK has code built into it for Final Validation testing and script generation. Instrumented so that when a debug flag (build define APDK_CAPTURE) is set, all API calls along with their data are recorded in what is called a script. From this script, a large class of problems can be found that would otherwise potentially take a very long time to debug.

Recording a Script

The first step in producing a script is to decide on the mechanism for storing it, and the format to be used. The easiest way to produce a script is to write the data to a file on local storage using standard "fopen/fwrite" commands. If such storage is available, the code included in the APDK will be sufficient. The data can be written in either binary or ASCII form; the binary form will produce smaller files and run faster.

Hewlett-Packard has encountered some systems where a storage disk is not available and the only means of capturing data is to use printf commands

that are sent to a terminal window. In this case the base-class scripting code has to be overridden (in the derivative **SystemServices** (p. 33) class); the code can be copied as is from the base-class routines, replacing every file-writing call (fwrite, fputc, etc.) with a display call (printf), and avoiding the fopen/fclose calls in the methods OpenDebugStreamR and CloseDebugStreamR.

In order to capture the data from the terminal window, the ASCII format must be used. Thus the code that needs to be overridden is in the AsciiScripter subclass of Scripter, found in the file script.cpp. The second step needed is for the calling code to invoke the **SystemServices** (p. 33) methods InitScript and EndScript. InitScript, as usual, should be called after the **SystemServices** (p. 33) object is constructed and before any other API calls are made. InitScript takes two parameters: the name of the script file to be created on disk (ignored in the diskless implementation), and a flag that is TRUE if ASCII format is to be used. EndScript should be called after everything else has finished and been cleaned up, except of course for destruction of the **SystemServices** (p. 33) object. Both these calls should be conditionally compiled into the calling code, so that they only occur in debug builds (see next paragraph).

The final step is to compile all the code with the symbol APDK_CAPTURE defined, and with the files in the "debug" subdirectory included in the build. Note that APDK_CAPTURE should not be defined in any production builds, as performance will be severely degraded. Run the test print-job and the resulting file can be used for debuging.

9.18 Frequently Asked Questions

9.18.1 APDK Program FAQ's

Q: Can developers receive technical support for APDK?

A: HP no longer provides any support for developing or debugging APDK based drivers.

Q: Can developers submit hardware or software products to HP for testing?

A: Developers bear full responsibility for qualifying their own hardware and software with HP printers. All types of Support for APDK from HP are discontinued.

Q: Does HP offer solutions for customers who would like to print-enable their Linux operating systems?

A: Hewlett Packard is pleased to offer the HP Linux InkJet Driver as a add-on to the GNU Ghostscript application. The source code for this driver and associated support for it is available at no charge from `http://hplipopensource.com` This HP driver uses GPL Ghostscript to

convert PostScript into print rasters that are passed through shared memory to our driver which executes as a server. A Ghostscript update kit is included with the HP solution.

The above mentioned HP Linux InkJet Driver and corresponding support for it should not to be confused with the HP Appliance Printing Development Kit (APDK). The APDK is a small foot print limited functionality driver intended for non-windows applications and devices, and is available as open source code from `https://spp.austin.hp.com/SPP/SPPMigrated/apdk.aspx`. The HP Linux InkJet Driver is based on the APDK core code but contains driver code and extentions to the APDK to take advantage of the Linux environment.

The following are questions that have been asked about the Hewlett-Packard Appliance Printing Software Development Kit (APDK).

9.18.2 Compiling

Q: What defines are usually turned on when you compile?

A: The defines are covered in Building Your Driver in the Getting Started document.

Q: I need the platform.h file to compile, but I can't find it. Where is it?

A: You, the developer, must create the platform.h file. A platform.h.readme file with further instructions has been included in the zipped source code portion of the APDK.

Q: My system does not have a C++ compiler. Do you have a C version of the APDK?

A: The minimum software requirement for using the APDK is a C++ compiler and linker as well as Standard libraries supporting string and other functions. A C++ cross compiler may be used to generate binary code for your system. In such a case you would need to move the binary and the C interface .h file over and your code can then use the C only interface. The C interface is in the file hpprint_c_api.h and the implementation to hook up your glue code to the APDK binary is in hpprint_c_api.cpp.

9.18.3 General APDK FAQ's

Q: What features does my printer support?

A: A list of the features supported by printers can be found in **Printers and Features Supported** (p. 86) documentation. In order to keep the printer driver code size small, not all features that come with a specific printer are included in the APDK code.

Q: What printers does the APDK support?

A: The list of supported printers can be found in **APDK Supported Printers** (p. 84). Any printers not listed in this table are most likely not supported. Printers introduced to the market after the latest APDK release should also be considered not supported. Printers that are not Hewlett-Packard printers, or are not Hewlett-Packard DeskJet printers (such as Hewlett-Packard Laser Jets, Office Jets or InkJets,) are also not supported unless listed.

Q: Which printer should I choose?

A: If code size is not a factor, Hewlett-Packard recommends developers utilize the monolithic APDK code set that implements all APDK-supported DeskJet printers. However, if code size is a factor, developers should choose a printer that takes into account the host system hardware environment and end-user requirements. If your host system has USB, be sure to choose a printer that has USB capabilities. Conceptualize your customers' printing requirements including color and image quality. Determine the price your customers will be willing to pay for a printer. Refer to the Printers Family / Functionality Table below to determine which printers meet the needs of your host system and your end users.

Q: Why am I not able to use all the features that come with my printer?

A: While this APDK code does contain space-optimized versions of many of the same factors as full-featured HP windows drivers, not all printer features could be included and still meet the small footprint required by many non-pc host systems. Hewlett-Packard has slim-lined this code-base to meet all the needs of standard printing functions, while foregoing printer "options" not included with all printers.

Q: What levels of PCL do DeskJet printers support?

A: DeskJet printers support PCL level 3 plus extensions.

Q: What makes DeskJet 600, 842c, 932c, and 952c printers "iTV Compatible"?

A: These four printers are being launched in conjunction with the AT&T interactive digital cable service which means the AT&T platform (O/S and Software) supports the printers. In general, the majority of currently shipping DeskJet printers are capable of being supported by this APDK. For a complete list of APDK-supported Hewlett-Packard DeskJet printers refer to the Printing document's Printer Family / Functionality Table.

Q: Is there support for built-in character sets other than Latin 1 for devices being deployed outside the US?

A: There is no support for built-in character sets other than Latin 1. (Latin 1 is not designed to contain every European accented character.)

Q: What is the difference between fonts and character sets?

A: Character sets define the character mappings, whereas fonts define the look of the character.

Q: I need greater detail about the DeviceId string so I can explore the possibility of adding more functionality. Can you help?

A: The interpretation of the DeviceId string is basically intended for the APDK core code and that is the reason for it being supported as documented. Aside from the implementation of **SystemServices** (p. 33), we expect partners to only call functions in the APDK public interface, and do not expect anyone to add functionality. As new functionality becomes available, it will appear in the public interface.

Q: Does HP offer solutions for customers who would like to print-enable their Linux operating systems?

A: Hewlett Packard is pleased to offer the HP Linux InkJet Driver as a add-on to the GNU Ghostscript application. The source code for this driver and associated support for it is available at no charge from `http://hplipopensource.com`.

The above mentioned HP Linux InkJet Driver and corresponding support for it should not to be confused with the HP Appliance Printing Development Kit (APDK). The APDK is a small foot print limited functionality driver intended for non-windows applications and devices, and is available as open source code from `https://spp.austin.hp.com/SPP/SPPMigrated/apdk.aspx`. The HP Linux InkJet Driver is based on the APDK core code but contains driver code and extentions to the APDK to take advantage of the Linux envronment.

9.18.4 Data Flow and Hardware Functionality

Q: Is it okay that a bunch of NULL bytes went to the printer as soon as the job was initiated?

A: Yes, this is used to flush any data that may have been in the printer buffer.

Q: Can you describe the supported I/O protocols for Hewlett-Packard printers?

A: A discussion surrounding supported IO protocols is included in the Low Level I/O Support document.

Q: What is Raster data as it relates to printing?

A: Images are represented digitally as two-dimensional grids of colored dots or pixels. Each pixel is represented by some combination of color values. The representational pixel format accepted by the APDK code is RGB, meaning one byte (a value from 0 to 255) for the red component,

followed by one byte (again, a value from 0 to 255) for the green component, and one byte for the blue component thus 3 bytes represent a pixel in RGB format. Example: a shade of blue could be written as (00,00,11).

A raster is a single row of pixels. If there are N pixels in a row, then one unit of raster data is 3 ∗N bytes, starting with 3 bytes for the leftmost pixel, 3 for the next pixel, and so on. (Refer to the Glossary for a global definition of "raster".)

Q: Everything works fine except that some printers don't appear to catch the out-of-paper or paper jam condition.

A: Some printers, namely the 640c and Apollo-branded printers, do not update the status-byte when used with USB. Therefore, the APDK core code error handling cannot interpret the error. If the user notices the out-of-paper error on the printer (blinking paper light) and handles this condition, the job will continue. Otherwise the print job will time-out with a 'Communication Error' after several minutes.

Q: I can print to the printer fine until I try to read the device ID or status byte during the job. What's going on?

A: Some early printers, released when USB was still a cutting-edge technology, tended to exhibit issues with deviceID/status byte requests when the printer's buffer was full. A full buffer is a common occurrence during the course of a print-job and can be forced with an out-of-paper condition or by opening the top cover. The APDK core code specifically handles these printers (640/810/830/880/895) internally by 'turning off' it's status checking after discovering one of these printers. However, it has been discovered that some implementers of the USB Printer Class Driver (the interface by which the APDK or any other printer driver communicate over USB to the printer) incorporate their own deviceID or status byte checks during the course of a print job. Namely, Linux kernels previous to 2.4.0 performed this action and consequently exposed the issue. It is our recommendation that the USB Printer Class Driver not perform these device ID or status byte checks until the printer driver makes the request.

Q: I can't communicate with the printer using USB.

A: This could be a problem where the USB Printer Class Driver is attempting to perform a SOFT_RESET during the course of initializing the printer (when the printer is power-cycled or is attached to the host). As with device ID and status byte requests, SOFT_RESET is a command the Class Driver is required to implement as part of the 'USB Printer Class Driver Specification'. Some implementers of the Class Driver have chosen to internally make this request to the printer - which is NOT a requirement of the specification. Further background on this issue is that this stems from an inconsistency in the USB Specification itself and an actual change in the Printer Class Driver Specification between versions 1.0 and 1.1. This results in a confusion of expected hand-shaking behavior between the Printer Class Driver and the printer. It is our recommendation

that the USB Printer Class Driver not perform a SOFT_RESET request of the printer unless instructed by the printer driver.

Q: I can get a device ID from the printer, so I know it's there, but it won't take any data.

A: This is again an issue with the USB Printer Class Driver. In this particular case, a non-functional interface protocol has been opened for communication. Depending on core USB implementation, the printer may actually appear to take up to 256 bytes before failing - this is just the result of filling an internal FIFO on the host.

During printer instantiation, the Class Driver will walk down through a list of structures (device/config/interface/endpoint) to establish communication with the printer. DeskJet printers, in compliance with the USB Printer Class Driver Specification, have multiple 'alternate' interfaces available. Unfortunately, the 'uni-directional' interface, which has a bInterfaceProtocol=01h in it's interface descriptor is non-functional. The interface with functional endpoints has a bInterfaceProtocol=02h (bi-directional). The recommended method for determining which of the alternate interfaces to SET is to walk through each of the alternate interfaces until the descriptor with the fore mentioned 'bInterfaceProtocol=02h' is found. This logic is required because different printers will return these interfaces in different order, making a solution of "use alternate 0" or "use alternate 1" impossible as a general solution.

Also note, the endpoint numbers can flip around based on which interface has been opened or which printer is used. Do not assume that the first endpoint is bulk OUT on all printers, for instance. The endpoint descriptor's bEndpointAddress uses bit7 as defined by the USB Printer Class Driver Specification to indicate bulk IN vs bulk OUT connectivity. This must be checked on a case by case basis as a printer is attached to the system and instantiated by the class driver.

Q: Obtaining the USB Printer Class Driver.

A: The USB Printer Class Driver is an external driver component supplied by the host O/S. If you are encountering one of the above errors and you do not have access to the Class Driver code, please contact your O/S system integrator to alert them of this issue.

9.18.5 Additional Required Reading Materials

Q: Why do I need the IEEE Specifications?

A: IEEE Specifications describe the method for implementing low-level I/O operations. Some examples of low-level I/O are a request for the device ID, and a request for a 1284 status byte, both of which are needed and expected by the printer driver. The document also describes negotiation into different modes, such as NIBBLE, COMPATIBILITY, and ECP.

Q: How do I get them?

A: Due to copyright laws, developers will need to purchase their own copies of the IEEE Specifications. The phone number for ordering IEEE publications is (800) 678-IEEE (US), 8AM-4:3O PM (EST). Outside of the US and Canada, the number is (908) 981-1393. To order by fax, dial (908) 981-9667.

The online store is located at shop.ieee.org/store. The user can also search on any part of the title, which is "IEEE Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers." A search for "bi-directional" will achieve the same results.

9.19 Glossary

This glossary consists of terms found throughout the Hewlett-Packard Appliance Printing Software Development Kit.

A

Application Programming Interface (API) - The set of function calls by means of which a software library is accessed, (i.e., **PrintContext** (p. 22).)

B

Bit - A binary digit that is 0 or 1.

BOOL - A data type of true or false, denoted by an integer, typically 1 or 0.

Byte - A sequence of 8 bits used to encode data.

C

Color filtering A method of coalescing similar colors in a raster to improve compression. **Consistency** - An IQ attribute that refers to the degree to which quality attributes and defects are uniform throughout a print job.

D

Data Member (of a class or an object) - a variable (or constant) that belongs to the public interface of that class/object. See also, Method (of a class or an object).

Downsample - To select a subset of a given image's pixels (say every Nth pixel) of dimensions [X x Y], so as to produce a smaller version of the same image with dimensions [X/N x Y/N]. (See also, Upsample.)

Dots Per Inch (DPI) - The number of dots of color in a square inch. Refers to the device dot placement addressability.

E

ECP (Extended Capabilities Port) - A term used in IEEE Specifications that refers to a mode that supports bidirectional communication. The APDK code only supports ECP forward mode. (See also, RLE.)

F

Function - any callable code that returns a value as its result.

G

Glyph - A picture of a character.

Graphic User Interface (GUI) - Program commands (code) in the form of buttons, dialog boxes, menus, or icons that make interaction between the user and the program easier.

H

Hook - Another word for interface or the API.

Host System - The system, i.e. the operating system together with relevant services (I/O, user-interface, etc.) and relevant applications (browser or other applications that print), in which the driver is embedded, and on which it depends.

I

Integration Testing - Testing to find failures in interfaces between subsystems, modules, and programs. This test type is needed primarily when a new module or subsystem is being added in. The most productive testing usually starts with a technical knowledge of the interfaces between the modules. Some regression testing of the module may be included.

Interface - A collection of related methods or functions working together to provide some functionality.

IQ - The abbreviation for image quality. The clarity of the writing device's output.

I/O - Input/Output. For this APDK, I/O refers to the calling and retrieving of low-level information (such as printer type and specs) between the printer driver and the host system.

J

K

KCMY - Abbreviation for Black, Cyan, Magenta, and Yellow. (See also RGB, sRGB.)

L

Landscape - The orientation of physical page rotation, either 90 or 270 degrees.

Logical Page - The actual page as defined by the application. The Printer Control Language's logical page (or addressable area) defines the area in which the cursor can be positioned. Logical page size may be changed by PCL commands. It can be larger than the printable area, which is determined by the technology of the printing device.

M

Media - The substrate that is printed upon, such as paper, glossy paper, photo paper, envelopes, or transparency film. Not all medias are accepted by this APDK.

Media Axis - The axis along which the media moves.

Media Size - See Paper Size.

Media Source - The location, or source of the print media. "Where is the print media being pulled from?" This is usually expressed as Tray1, Tray2, Manual feed, or Manual Envelope.

Method (of a class or an object) - A function that belongs to the public interface of that class/object. See also, Data Member (of a class or an object).

N

O

Orientation - Either portrait or landscape, often defined in degrees of rotation (0, 90, 180, or 270).

Operating System (OS) - Example: Windows, Unix, Linux, or Macintosh.

Overloading - Two methods within a class with the same name, but with different parameters, even though different outputs can occur. Parameter lists determine which is to be used (i.e., **Font** (p. 18)).

P

Page Description Language (PDL) - Proprietary language containing the DDI commands that describe the contents of a logical page.

Paper, Plain - The term loosely used to refer to standard office papers such as bond and copier paper. In reality, there is no such thing as plain paper since every paper is designed for some specific use.

Paper Size - The size of the media being printed on. Example, Letter, Legal, A4, or Envelope.

Paper, Special - Paper or other media (such as Hewlett-Packard Premium Glossy Paper) which has been specifically formulated or processed for optimal use with a particular printing or writing system or in a particular print mode.

Paper Type - The type of media being printed on. Example: plain paper, bond, special, glossy or transparencies. See also, Media.

Pen - A print cartridge. Within the ink jet industry, print cartridges are typically called pens. However, consumers known them as print cartridges. The two terms are synonymous.

Peripheral - Any item connected to the host system. Example: mouse, keyboard, scanner, or printer.

Physical Page - The actual sheet of paper or media - what the user thinks of as a "page". Physical page size is determined by the size of the media inserted in the printer.

Pitch - A measurement defined in terms of characters per inch'. A pitch of 10 (or 10-pitch) means that 10 characters (at the specified point size) will cover 1 inch across the page.

Pixel - In this document, one or more logically addressed dots. Examples are: multiple black dots to achieve gray scale, or one or more color dots in the same location to achieve a desired color.

Pixel-rows - A row of pixels. Each pixel is input to the APDK code set as 24-bit RGB value. The number of pixels on one row is set by the client in the **PrintContext** (p. 22) API.

Point size - Only used in relation to fonts, the point size of a font corresponds to the amount of vertical space required to store the tallest character in the font set. One point is technically 1/72".

Portrait Orientation - The orientation of physical page rotation, either 0 or 180 degrees.

Print Mode - An imaging algorithm that specifies dot resolution, carriage velocity, and other printing specifications in order to optimize image quality, throughput, and media compatibility. Best, Normal, and Econo-Fast are examples of print modes.

Print Quality - A user-specified setting for the quality of the printed output, usually expressed in terms of print modes: EconoFast, Normal or Best.

Printable Page - See Printable_region.

Printable Region - Refers to the area of the printable page in which printable output may occur. For example: the default printable region on standard letter size paper (8.5 x 11") is 8 x 10". This allows 0.25" left and right margins, respectively, as well as a 0.33" top margin and 0.67" bottom margin.

Printer Driver - A part of the printing system that interfaces with the operating system, and generates printer ready data. The APDK is not a printer driver, but does contain core code needed to create a printer driver.

Printer Ready Data (PRD) - Pre-formatted data in the language of the target printer, which needs no processing before being sent to printer.

Q

R

Raster Data - Raw data for a print head, detailing print information made up of dots in a horizontal row, or dots which are arranged in rows and columns, ending when the print head completes a carriage return.

Raster Row - One line of consecutive pixels in the scan (x-axis) direction.. **RGB** - A 24-bit value representing a color for one pixel (or the format for representing a pixel), with 8 bits each for Red, Green and Blue. (See sRGB.)

RLE (Run Length Encoding) - A term used in IEEE Specifications that refers to a compression mode that may exist in ECP transfers. This protocol is not supported by the APDK code. (See also, ECP.)

S

Sharpness - An IQ attribute that refers to the distinctness of the edge transition at an object's boundary.

sRGB - An industry standard color space that enables different color-capable devices (e.g. displays, printers, etc.) to exchange color information in a way that helps maintain color accuracy. This enables the colors

that the user sees on the screen to more closely match the colors that are printed.

System Testing - System testing looks for defects that belong to the system as a whole.

T

Transient Data - Data that exists only during the processing of a print job. The majority of this data is generated by user settings for the print job, and information that is generated as part of processing the print job.

U

Upsample - When an image is given at a lower resolution than that of the printer, it must be expanded or scaled up by interpolating between neighboring pixels to produce enough pixels to fill the target region. The interpolation may be "dumb", i.e. a simple averaging of neighboring pixel values, or it may be "smart", as in Hewlett-Packard's proprietary SmartFocus technology. (See Scaler object in the API.) (See also, Downsample.)

V

Variable Member (of a class object) - a variable that belongs to the interface of that class/object. **Visual Comfort** - An IQ attribute that refers to the degree to which an image does not cause a viewer visual fatigue or distress.

W

X

Y

Z

Index